Towards Location-aware News Feeds and Recommendations

A THESIS
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Jie Bao

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
Doctor OF Philosophy

Prof. Mohamed F. Mokbel

June, 2014

© Jie Bao 2014
ALL RIGHTS RESERVED

Acknowledgements

First and foremost, I would like to thank my advisor, Prof. Mohamed Mokbel, for his great help and support. Prof. Mokbel guided me to understand how the high quality research is done. During my Ph.D career, he always encourage me to "go deep" in thinking problems, and have a "system-oriented" mind in doing research projects. These two instructions have been proven very effective. I would like to thank him for so many hours of discussion and insightful feedback.

I would like to thank Prof. Shashi Shekhar, Prof. Jaideep Srivastava, and Prof. Gediminas Adomavicius for serving on my thesis committee. Their feedback and guidance have been absolutely invaluable.

As a member of Data Management Lab, I would like to thank my fellow lab mates (in alphabetic order), Louai AlArabi, James Avery, Ahmed Eldawy, Abdeltawab Hendawi, Mohamed Khalefa, Justin Levandoski, Amr Magdy and Mohamed Sarwat, for all the insightful discussions and feedback. Especially, I want to thank Chi-Yin Chow for the tremendous help and guidance during the first and second years of my Ph.D career.

Also, I would like to thank all my friends in University of Minnesota (in alphabetic order), Bin Cao, Yanhua Li, Zhe Jiang, Xiaofan Wu, Wei Zhang, and Xun Zhou for all helps during the five years here.

Moreover, I would like to acknowledge those industry partners that supported this work, Dr. Yu Zheng, Dr. Xing Xie, and Dr. Jing Yuan from Microsoft Research Asia, as well as Dr. Chandra Narayanaswami, Dr. Scott MacFadden, and Dr. Ajay Deshpande from IBM T.J. Watson Lab.

The work reported in this dissertation was supported in part by NSF Grants IIS-0811998, IIS-0811935, CNS-0708604, and IIS-0952977

Dedication

To my parents, *Jianqiang Bao* & *Zheming Ni*, my wife, *Xiang Zhang*, and my son, *Yuncheng J. Bao*.

Abstract

With the advances in positioning techniques, such as GPS, cell-towers and WiFi, users can enjoy location-based services more easily than ever before, e.g., on their smart phones and GPS devices. However, with the popularity of recommendation (e.g., Amazon & Netflix) and socialization (like Facebook and Twitter) functionalities in the web services, users of location-based services are no longer satisfied with the static results returned from a spatial database. At the same time, more and more spatial information, such as geo-tagged photos and check-ins are generated in the traditional social networking services. As a result, users are calling for the next generation of location-based service, i.e., location aware news feed and recommendations, which can provide the user with the more personalized and socialized services.

In this thesis, I present my vision of the next generation of location-aware service, which enables the social networking services with location awareness. First of all, in this thesis, I present the unique properties that location information brings to the traditional social networking and recommendation services. After that, I summarize the potential challenges in building efficient and effective location-aware news feed and recommendation services from both the system and user's perspectives. Then, I present a prototype system, (i.e., Sindbad, a location-aware social networking system), to demonstrate three main services provided in the system, as: 1) location-aware news feed service, which efficiently returns the user with spatial-aware messages from her subscribed friends. 2) location-aware news ranking services, which provide a set of efficient news ranking and message updating algorithms for the user to get more relevant news, based on her social-spatial preferences. And 3) location-aware recommendations, which provides suggestions based on the social knowledge from the local experts and the preferences mined from a user's location history. Finally, the thesis is concluded with the overall contributions and some potential further research directions.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

With the advances in positioning techniques, such as GPS, cell-towers and WiFi, users can enjoy location-based services more easily than ever before, e.g., on their smart phones and GPS devices. As a result, location-based services have been very popular. Most of the location-based services can be abstracted as different types of spatial queries to a POI database or the road networks, for examples, 1) *spatial range query*, which may find the nearby restaurants in a given spatial range from the users, 2) *k-nearest neighbor query*, which may find the nearest k gas stations from the user's current location, 3) *shorest path query*, which can provide the user with the most convenient route to her destination.

However, with the popularity of recommendation (e.g., Amazon & Netflix) and socialization (like Facebook and Twitter) functionalities in the web services, the term Web 2.0 becomes very popular, which is associated with web applications that facilitate participatory information sharing, crowd-sourcing, interoperability, user-centered design, and collaboration on the World Wide Web [1]. The popularity of Web 2.0 came about as a direct result of the wide increase of web-based user-generated content and social networking technologies. In Web 2.0, the World Wide Web has moved from being an interface for information retrieval to an interactive medium where users can share information, upload user generated content, and interact with other users. Following the success of Web 2.0, a flurry of 2.0s have appeared including Library 2.0 [2], Travel 2.0 [3], Government 2.0 [4], and even Revolution 2.0 [5]. All application of 2.0s rely mainly on social interaction among participants, where the knowledge of one person helps others in an information participatory media.

As a result, users are not satisfied with the results returned by the current location-based services and looking for more socialized and personalized location-based service, i.e., location-based service 2.0. The main reason behind the requirement comes two folds: 1) *location-based service needs more social knowledge.* For example, when a user wants to find a restaurant in the location-based service, the real question asked by the user may not be finding the nearest one, but the best one for her [6], which may need the consideration of different social knowledge, such as the cuisine style, price, waiting time, user ratings/comments and etc. And 2) *the current social networking/recommendation services need to deal with more spatial information.* For example, there are more Geo-tagged information appeared in the traditional social networking services, like Facebook Place and Twitter Nearby [7]. On the other hand, the traditional recommendation services, like Netflix or Amazon currently pay more attention on the user's neighborhood and will recommend to the user with movies or other products that nearby people have liked, e.g., Amazon local. Both of these application scenarios call for the more socialized and personalized location-based services. To this end, in this thesis, I developed three main services to extend three of the most popular social services, i.e., news feed, news ranking and recommendation with location awareness.

1.2 Properties of Location Information

Location information brings the following three unique properties to the traditional online social networks, as shown in Figure 1.1,:

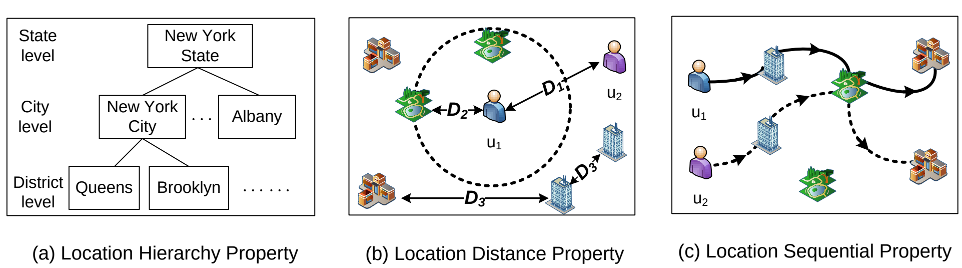


Figure 1.1: Unique Properties of Locations.

Hierarchical. Locations span multiple scales: for example, a location can be as small as a restaurant or as big as a city. Locations with different granularities form a hierarchy, where locations on a lower tiers refer to smaller geographic areas. For example, a restaurant belongs to a neighborhood, the neighborhood belongs to a city, the city belongs to a county, and so on (see Figure 1.1a). Different levels of location granularity imply different location-location graphs and user-location graphs, even given the same location histories of users. These hierarchical relationships need to be considered as, for example, users who share locations at a lower level (such as a restaurant) likely have a stronger connection than those who share locations at a higher level (such as living in the same city). This hierarchical property is unique in LBSNs, as it does not hold in an academic social network, where a conference never belongs to others.

Measurable Distances. Connecting the physical world to a LBSN leads to three new geospatial distance relations, the distance between different users' locations (shown as D1 in Figure 1.1b), the distance between a user and a location (shown as D2 in Figure 1.1b), and the distance between two locations (shown as D3 in Figure 1.1b). According to the first law of geography posed by Waldo Tobler [8], "*everything is related to everything else, but near things are more related than distant things*", we propose that distance affects an LBSN in the following three ways. 1) The user-user distance influences the similarity between users. For example, users with a history of visiting nearby locations are more likely to have similar interests and preferences [9, 10], and users who live close to each other are more likely to be friends [11]. 2) The user-location distance influences the likelihood a user will be interested in a location. For instance, users in Foursquare visit restaurants close to their homes more frequently than others [12]. 3) The location-location distance affects the correlations between locations. For example, shopping malls are often placed close to each other [13].

Sequential ordering. Subsequent visits by a user to two locations creates a relation with a chronological ordering. For instance, the two users in Figure 1.1c share a location visiting pattern. From the time of each visit, we can create an ordering which may indicate some similarities between their preferences [14] or may imply traffic conditions [15].

1.3 Challenges

In this section, we present the challenges in developing the location-based services 2.0 from two different perspectives: 1) System perspective and 2) User perspective.

1.3.1 System Perspective

From the system perspective, in a socialized location-based services, the most important issue is to provide the responsive service to its users. As a results, it presents two main challenges here, as: 1) efficient geo-social information access techniques, and 2) an efficient geo-social information update techniques.

Providing location-based services in a social network setting requires more effort than the traditional social networking service, as it requires an additional step to filter the irrelevant information based on the user's location. For example, in a location-aware news feed application, which returns the spatial relevant messages from a user's friends, the system not only needs to retrieve the new items based on the user's social relation, but also needs to filter the retrieved items based on the user's location. If we simply applies an spatial filter on top of the existing service, the user would experience a significant response delay. Also, on the other hand, the system essentially retrieves much more unnecessary information that will be filtered based on the user location and may be overwhelmed by the massive request from users. To this end, we need to apply more spatial pruning techniques to minimize the redundant computations, for example, building spatial indexes over the geo-spatial dataset and designing more efficient accessing algorithms that push the spatial operator deep inside to improve the system efficiency.

A socialized location-based service also requires information updating technique, which is usually overlooked in a traditional location-based service (where most of the information like road networks and POIs are static). To utilize the user-generated geo-social content and reflect the social opinions in the location-based service, the system may face with the more frequent updates from the user, including the geo-tagged photos, comments and ratings, than the traditional social networking/recommendation services. For example, in traditional recommendation services, like Amazon and Netflix, a user may watch one or two movies or purchase three items a day. However, in a socialized location-based service, a user can check-in multiple venues or upload tens of geo-tagged photos in a day. To this end, we need to choose an update-friendly index to minimize the system overhead. Also, more efficient recommendation models may be needed to reduce the cost in including the new opinions/ratings from the users.

1.3.2 User Perspective

From a user's perspective, location-based service 2.0 should not only provide responsive services, but also need to be more effective, which means that the users require more relevant information based on her current location, preferences and the other people's opinions. As a result, a socialized location-based service needs to take consideration of three main factors: 1) a user's current location, 2) a user's location history, and 3) other users' location histories.

A user's current location. First of all, the spatial distance property implies that people are more likely to visit nearby locations than distant ones or more interested in the things happened nearby. Secondly, given the user's current location, it indicates a spatial constraint for generating recommendations, but also influences user preferences. For example, beaches might be given a high recommendation rank to a user traveling to Hawaii, even though the user prefers sporting events more than beaches typically. The same user may be more interested in seeing the status of her friends living in Hawaii.

A user's location history. Earlier works, e.g., [16, 17], have suggest that a user's historical behaviors is a powerful indicator of the user's preferences. A user's accumulated location history (e.g., check-ins and geo-tagged photos) reflect more accurately a user's experiences, living patterns, preferences and interests than the user's online behaviors [18]. However, it is non-trivial to model a user's location history due to the hierarchy, distance, and sequential properties of locations. Moreover, learning a user's personal preferences from the user's location history is very challenging for the following reasons. 1) As users do not share their locations everywhere, a full set of a user's location history does not exist. Learning a user's preferences from sparse location data is challenging. 2) A user's preferences span multiple kinds of interests, such as shopping, cycling, and arts, rather than consisting of binary decisions, e.g., a set of 'like or dislike' statements. 3) A user's preferences have hierarchies and granularity, such as "Food" $\rightarrow$ "Italian food" $\rightarrow$ "Italian pasta". 4) A user's preferences are constantly evolving (and location dependent).

Other user's location histories. Location histories generated by other users make up the social opinion, which is one of the most important information bases for the socialized location-based services, like making location recommendations. However, it is not an easy task to extract the social knowledge from the users location histories. For example, users have different degrees of knowledge about different geospatial regions. Moreover, the knowledge of a user is region-related and changes over the granularity of a location. A travel expert in New York City might have less knowledge of Seattle. Likewise, people who are shopping experts in one district of a city might not be the most knowledgeable of the city as a whole.

1.4 Sindbad System Overview

As a part of my thesis project, we, members of Data Management Lab, developed an online system demonstration, i.e., Sindbad [19][1] , to illustrate our vision about location-based service 2.0.

Users of Sindbad can entertain one or more of the following functionalities: (a) select their friend list as well as getting listed as friends to other users in a same way like traditional social network systems, (b) post (spatial) messages and/or rate (spatial) objects (e.g., restaurant or movies), which will be seen by their friends, (c) once a user logs on to Sindbad, the user will see an incoming location-aware news feed from the user friends. The news feed is selected based on both the user location and the spatial extents of the posted messages, and (d) get a location-aware recommendation about spatial items, e.g., restaurants, or non-spatial items, e.g., movies. The recommendation is based on the user location, the item location, and what are the items that the friends of the user have liked.

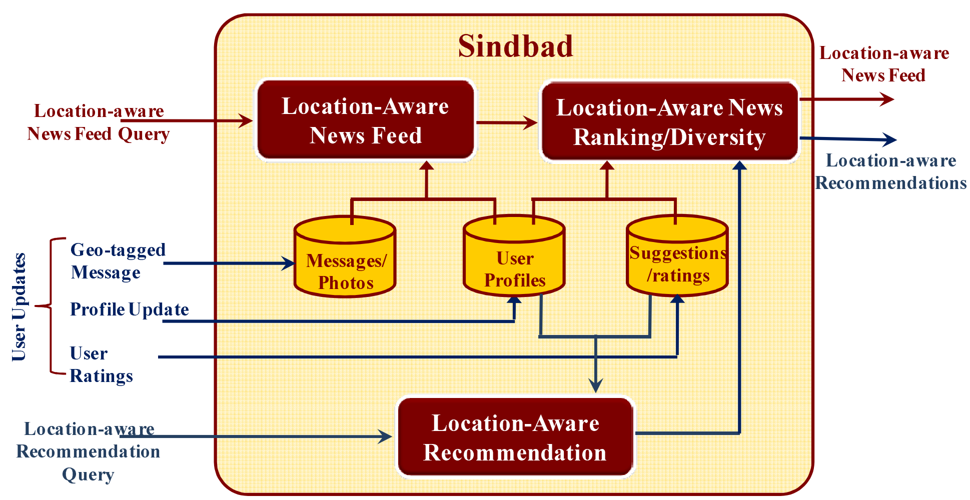


Figure 1.2: Sindbad System Overview.

[1] http://sindbad.cs.umn.edu

Figure 1.2 depicts the Sindbad system architecture that consists of three main modules, namely, location-aware news feed (i.e., GeoFeed in Chapter 3), location-aware news ranking (i.e., GeoRank in Chapter 4), and location-aware recommender (recommender service in Chapter 5), and three types of stored data, namely, spatial messages, user profiles, and spatial ratings. Sindbad can take five different types of input (i.e., through the API interface): profile updates, a new message, a new rating, a location-aware news feed query, and a location-aware recommender query. The actions taken by Sindbad for each input is described as follows:

Profile updates. As in typical social networking systems, Sindbad users can update their personal information, their friend list, or accept a friend invitation from others.

A new message. Users can post spatial messages to be seen by their friends, if relevant. A spatial message is represented by the tuple: (MessageID, Content, Timestamp, Spatial), where MessageID and Content represent the message identifier and contents, respectively, Timestamp is the time the message is generated, while Spatial indicates the spatial range for which the message is effective. The message is deemed relevant to only those users who are located within its spatial range.

A new rating. Sindbad users can give location-aware (spatial) ratings to various items in a scale from one to five. Location-aware (spatial) ratings can take any of these three forms: (1) Spatial ratings for non-spatial items, represented as a four-tuple (user, user-Location, rating, item); for example, a user located at home rating a book, (2) Non-spatial ratings for spatial items, represented as a four-tuple (user, rating, item, item Location); for example, a user with unknown location rating a restaurant with an inherent location, and (3) Spatial ratings for spatial items, represented as a five tuple (user, userLocation, rating, item, itemLocation); for example, a user at his/her office rating a restaurant with an inherent location.

Location-aware news feed queries. Once a Sindbad user logs on to the system, a location-aware news feed query is fired to retrieve the relevant news feed, i.e., messages posted by the user's friends that have spatial extents covering the location of the requesting user. Details of the execution of the location-aware news feed query will be discussed in GeoFeed.

Location-aware recommendation queries. Sindbad users can request recommendations of either spatial items (e.g., restaurants, stores) or non-spatial items (e.g., movies) by explicitly issuing a location-aware recommendation query. The location-aware recommender module suggests a set of items based on: the user location, the item locations, and user preferences. Details of the location-aware recommender service will be presented in Chapter 5.

Location-aware ranking query. The results returned by the location-aware news feed and recommendation will be processed further by the location-aware ranking module to get only the top-k news ranking based on the spatial and social relevance. Details of the location-aware ranking module will be described in GeoRank in Chapter 4.

1.5 Road Map

In this thesis, we present our vision of Location-based Services (LBS) 2.0, where users can generate significant location-based content and enjoy meaningful location-aware interaction with both the system and other users. In other words, our approach is similar to the story of spatial databases over the last two decades, where the spatial functionalities were embedded inside existing database systems, making use of the existing infrastructure including query operators, optimizers, indexing, and transaction processing.

- Chapter 1 introduces the background and potential challenges of socialized location-based services.
- Chapter 2 briefly presents the related works for the different services proposed in this thesis.
- In Chapter 3 gives a detailed description of GeoRank system, which efficiently provides location-aware news feed for its users.
- Chapter 4 describes the GeoRank system, which provides the location-aware news feed ranking functionality for the users.
- Chapter 5 presents a novel preference aware location-based recommendation services, which studies a user's historical check-ins and provide relevant location recommendations for her.
- Chapter 6 concludes the thesis and presents some potential future research directions.

Chapter 2

Related Works

In this chapter, we discuss the state-of-art solutions that are related to our proposed systems as, 1) location-aware news feed, 2) location-aware news ranking, and 3) location-aware recommendation.

2.1 Location-Aware News Feed

This section highlights related work to GeoFeed, the location-aware news feed services, in two main areas; traditional news feed systems and location-aware social networks.

Traditional News feed systems. Most of existing news feed systems work in a similar way to publich/subscribe services, e.g., [20, 21, 22], which use a push approach to fan out the message notices to all their users. However, such systems are not applicable to address the location-aware news feed, as (a) they do not consider the spatial relevance of each message, and (b) using the push approach does not scale up for large number of publishers and subscribers as it is the case for social networks. For commercial systems, our only knowledge is about the Feeding Frenzy system [23] from Yahoo!, which we consider as our closest work and compare with it. The main idea of Feeding Frenzy is to build a cost model for deciding upon using the *pull* or *push* approaches as means for retrieving the news feed for a registered user. The only way to use the Feeding Frenzy system for the location-aware news feed problem is to attach a wrapper around it to filter any spatially irrelevant message from the users' news feed. However, that would be very inefficient as the spatial filter is applied as an afterthought solution. Our proposed system, GeoFeed, distinguishes itself from Feeding Frenzy in that it is built with the location-awareness

Location-based Social Network	Location Tag	Range Queries	Spatial Messages
Facebook Places [24]	√		
Renren [25]	√		
Sina Weibo [26]	√		
Loopt [27]	√	√	
Google Buzz [28]	√	√	
Foursquare [29]	√	√	
Twinkle [30]	√	√	
GeoFeed	√	√	√

Table 2.1: A Taxonomy of Location-based Social Networks

functionality in mind. Thus, the query evaluation methods, the cost model, and decision model take into account the spatial aspect of each posted message along with the location of each user.

Location-aware social networks. Existing commercial location-based social networks fall in two categories, as summarized in Table 2.1. The first category includes Facebook Places [24], Renren [25], and Sina Weibo [26], where they consider the location information of the message issuer as just an additional tag attached with the message. Then, a system user will get the same news feed (associated with location tags) regardless of the user location. The second category includes Loopt [27], Google Buzz Mobile [28], Foursquare [29], and Twinkle [30] where, in addition to having the location tags, they also give their users the ability to issue range queries to view the whereabouts of their friends. GeoFeed distinguishes itself from all these commercial products in two main aspects: (1) GeoFeed gives its users the ability to set the spatial validity range of each posted message., and hence give control to the message issuer to decide who should get the posted information. For example, the weather service provider may decide a tornado warning is relevant to followers located only in a certain area. (2) Unlike all existing systems that are built mainly to be used by mobile devices, GeoFeed offers a more flexible way for the users to share their geo-tagged messages. Users of GeoFeed can access their account in the same way they use Facebook, yet, they will retrieve more relevant location-aware news feed than that of Facebook users.

On the other side, related research prototypes in location-based social networks have either focused on: (a) message sharing [31, 32], where users can broadcast or receive public location-based messages, yet with no social awareness, i.e., there is no concept of friendship. Applying techniques form message sharing to the news feed problem is equivalent to having all queries evaluated with the (spatial) *pull* approach, which is very inefficient, (b) Privacy-aware search queries [33, 34, 35, 36], which enables private query search over users' friends, with no interest of the location-aware news feed functionality, or (c) location-aware recommender systems [37, 38, 39, 40], which suggests new places for their users. The functionality of recommender systems is fundamentally different from news feed systems, where the main goal is predict what the user would like rather than delivering the news feed from posted messages of users' friends.

2.2 Location-Aware News Ranking

This section highlights the related work to GeoRank, the location-aware news ranking services, in three main areas: (1) News ranking/news feed systems, (2) Efficient evaluation of top-k queries, and (3) Answer quality in top-k queries.

News ranking/news feed systems. Most of the existing news ranking systems, e.g., [41, 42], ranks the news by matching the content with the user's profile. Other systems, e.g., [43, 44], keep tracking of the users' clicking behaviors. Most recently, location information has drawn a significant attention in generating more relevant news, e.g., [45, 46], or enabling the location tagging, e.g., [31, 32]. Moreover, MobiFeed [47] provides the relevant news items based a user's predicated traveling path. However, these technique cannot be adapted in GeoRank directly, because all the above techniques assume the users are interested in the news items from all the sources. GeoRank, on the other hand, allows the users to get their news feed from the subscribed news sources.

To incorporate with the social awareness, news feed systems [23] have been proposed, whereas location-aware news feed systems [7, 19] further introduce the spatial relevance. However, these existing news feed systems limit their result as the most recent ones. At the same time, because of its important, several commercial products have also appeared to provide news feed services. However, none of them provides the similar service as GeoRank. For example, Facebook Places consider the location as just an additional tag, where their users get the same news feed regardless of user locations. FourSquare can only provide a view for the geo-tagged messages issued in a nearby venue, yet nothing about ranking the results based on a spatio-temporal function. GeoRank, on the other side, provides the users with a personalized top-k most relevant messages, considering both spatial and temporal proximity.

Efficient evaluation of top-k queries. The top-k query evaluation has been well studied in the literature, e.g., see [48, 49, 50, 51, 52]. All algorithms present extensions or variations from the famous TA algorithm that retrieves objects from a set of input lists, each ordered based on one attribute contributing to the overall ranking function [49, 50]. More recent algorithms have focused on supporting top-k queries on streaming environment and continuous queries, e.g., [53]. As mentioned in the previous section, existing algorithms are not applicable to GeoRank: (1) input lists have to be ordered on a contributing attribute to the ranking function. It is not a valid assumption in GeoRank, whereas the users issue news feed requests from different locations; (2) a top-k query is evaluated in an ad-hoc basis, which is not the case in GeoRank. Although the users issue news feed request for a unique set of sources, there are many cases they may share a part of sources [54]. Thus, shared execution techniques and index structures are needed to further optimize the system performance; (3) most of the existing techniques consider k is significantly higher than the number of input lists, which is completely the opposite in GeoRank, where k is significantly lower than the number of input lists. And (4) most of the existing top-k techniques overlook the updates in the input list. However, in GeoRank, new messages come continuously, where efficient updating method is also essential.

Answer quality in top-k queries. Several methods have been proposed to provide better quality of top-k queries that match the users' different requirements. Examples of such methods include but not limited to the skylines [55], hybrid multi-objective methods [56], and top-k dominance [57]. With the popularity of location-based services, spatial information has been introduced in the top-k rankings, e.g., k nearest neighbor queries [58], where only the distance proximity is considered. Then, spatial skylines [59] and spatial preference queries [60] incorporate other factors in the ranking function. Unfortunately, none of these techniques is directly applicable to GeoRank, as all of them assume that their input is stored in one table or index structure. This is not the case in GeoRank, as the top-k answer may be retrieved from different sources. In addition, GeoRank takes the social aspect, where the top-k messages have to come from the subscribed sources.

Most recently, recommendation techniques have been introduced [61, 39] in top-k rankings to utilize the user behaviors. The location recommendation is either based on mining the user's trajectories [39] or adjusting traditional collaborative filtering techniques to the spatial environment [61]. The latter relies on finding similar users, who do not have to be friends nor have to even know each other. This is different form GeoRank, where users only see the news from the sources of interest.

2.3 Location-Aware Recommendations

We summarize the existing location recommendations into two categories: 1) generic location recommendations and 2) personalized location recommendations.

2.3.1 Generic Location Recommendations

Regardless of the preferences of an individual, generic location recommendation systems encapsulate the public opinions on locations to provide people with the most popular venues or travel routes in a city. For example, [62] mines the most interesting locations and travel sequences from a large number of user-generated GPS trajectories. Given a user-location matrix, a HITS-based inference model was also proposed to predict the interest level of a physical location and the knowledge of a user. [63] further extends this work by considering the correlation between locations when doing the inference. However, both of them do not differentiate the locations from different categories. Though these recommendation systems have their own applications, sometimes, it would be difficult to say which one is more interesting, a shopping mall or a museum, as different users may have different answers.

2.3.2 Personalized Location Recommendation

Some simple personalized recommendation systems request a user to manually specify her personal interests by categories (like restaurants and parks) [64, 37], which will be employed to determine the POIs (around the user) to be shown on a mobile interface. As a user's preferences are not actually binary decisions and have a certain granularity, manually specifying personal preferences is obtrusive and usually bring a user too many or too few recommendations. Meanwhile, such systems do not incorporate other users' opinions on a venue, losing a lot of valuable information.

A branch of recent research starts learning a user's interests from the user's location history and incorporates the social environment of the user to make recommendations. Specifically, [65, 12, 13, 38] deposit people's location histories into a user-location matrix where a row corresponds to a user's location history and each column denotes a venue like a restaurant. Each entry in the matrix represents the number of visits of a particular user to a physical venue. Then, a user-based CF model is employed to infer a user's interest to an unvisited venue. However, the similarity between two users is simply represented by the Cosine similarity between the two users' rows, overlooking the features of human mobility in geographic spaces, such as sequential and hierarchical properties of locations. To better estimate the similarity between users, Zheng et al. [66] proposed a hierarchical-graph-based similarity measurement taking the human mobility features into account. The location recommendation system using the user similarity outperforms those using the Cosine similarity. While the user-based CF model is able to capture people's mobility in the physical world, it has a poor scalability as adding a new user into a system will trigger a large number of similarity computing operations. To address the problem of scalability, [67] proposed a location-based CF model using the location correlation mined from many users' GPS traces as a distance measure between two locations. The location-based CF model is slightly less effective than the user-based one while being much more efficient.

Unfortunately, solely using a CF model (no matter the user-based or the location-based) cannot handle the data sparseness problem very well if we directly formulate a user-location matrix. Though [68, 39] applied Single Value Decomposition to a user-location matrix so as to reduce the data sparseness problem to some extent, this method does not work well when there is no overlap between users' location histories. In fact, this is quite common when an individual travels to a city that is new to her.

Our recommendation system differs from the above-mentioned work in the following two aspects: 1) We project a user's location history into the category space and model a user's preferences using a WCH. This method handles the data sparseness problem and enables the computing of similarity between users who do not share any physical location histories, e.g., living in different cities. Unlike the traditional cold-start problem in the recommender system [69, 70], where the users or items come to the system with no ratings, a user is new only for the unfamiliar area in terms of the new city problem in location-based recommendation.

As we take advantage of the category information of the user's historical location, we can recommend locations to a user in a city based on her location history in other cities. 2) Pervious CF-model based methods have to infer a user's interests in a venue offline due to the heavy computation and then present the locations with a high ranking around a user. Such methods cannot guarantee the quality of the recommended locations as a user's current location is not truly incorporated in the inference. But, our system chooses candidate venues according to a user's current location (or any location specified by a user) and carries out the inference online. So, the venues recommended by our system are not only preference-aware but also really location-based.

Chapter 3

GeoFeed: A Location-Aware News Feed System

3.1 Introduction

Social networking systems, e.g., Facebook [24] and Twitter [71], and news aggregators, e.g., My Yahoo! [72] and iGoogle [73], are among the most popular web services nowadays. A common functionality shared by such web services is the *news feed* functionality, where users of social networks and news aggregators receive a set of news from their friends and favorite news sources, respectively. Due to the large volume of related news for each user, existing news feed systems opt to select a subset of k relevant news either based on the message timestamp, i.e., most recent k messages, or based on some diversity requirements. Unfortunately, such a selection ignores the spatial aspect of related messages, and hence, users may miss several important messages that are spatially related to them either because they are not so recent or do not satisfy diversity requirements. For example, when a traveling user logs on to a social network site, the user would like to get the news feed that match his/her new location, rather than sticking to the most recent news feed. The same concept can also be applied for users who keep logging on to the system from the same location, yet, they have a large number of friends. It is of essence for such users to limit their news feed to the ones related to their locations.

In this paper, we present GeoFeed; a location-aware news feed system that provides a new platform for its users to get spatially related message updates from either their friends or favorite news sources. GeoFeed complements the functionality of existing social networks and news aggregators to make them location-aware. Once a user u logs on to her favorite social network site that is equipped with GeoFeed, u will find the set of messages that are more relevant to her current location, e.g., a message about local news, a comment about a local store, or a status message targeting friends in a certain area. For a user u that has a set $\mathcal{F}_u$ of N friends (in a social network context) or follows a set $\mathcal{F}_u$ of N news sources (in a news aggregator context), GeoFeed abstracts the location-aware news feed problem to evaluating a set $\mathcal{Q}_u$ of N location-based queries posed by u. Each query $q_i \in \mathcal{Q}_u$ is posed to a friend $f_i \in \mathcal{F}_u$ to retrieve the set of messages that are issued by f_i and overlap with u's range of interest. u's range of interest could be the exact location of u, in which the location-based queries are point queries, or a range around u, in which location-based queries are range queries, e.g., get all the messages posed by my friends within r miles form my location. To limit the set of messages delivered to u, GeoFeed gets only k messages from each friend $f_i \in \mathcal{F}_u$. In the mean time, GeoFeed guarantees that each user u will get all the requested news feed within a response time threshold $\mathcal{T}_u$.

GeoFeed is equipped with three different approaches for evaluating each query $q_i \in \mathcal{Q}_u$, namely, (1) *spatial pull approach*, in which q_i is answered through exploiting a spatial index over the messages of friend f_i, (2) *spatial push approach*, in which q_i just retrieves the answer from a pre-computed materialized view maintained by friend f_i, and (3) *shared push approach*, in which the pre-computation and the materialized view maintenance at friend f_i are shared among multiple users that include u. Then, the main challenge of GeoFeed is to decide on when to use each of these three approaches to which queries. GeoFeed is equipped with an elegant decision model that decides about using these approaches in a way that: (a) minimizes the system overhead for delivering the location-aware news feed, and (b) guarantees a certain response time $\mathcal{T}_u$ for each user u to obtain the requested location-aware news feed. A better response time calls for using the *spatial push* approach for all queries, where all news feed are pre-computed. However, this results in a huge system overhead to maintain a massive number of materialized views, hence limit the scalability of the system to support more users. In contrast, favoring system overhead may result in evaluating more queries using the *spatial pull* approach as less views are maintained. However, users with large numbers of friends will suffer a significantly long delay when retrieving their news feed. GeoFeed takes these factors into account when deciding on which approach to use to evaluate each query q_i in a way that minimizes the system overhead, i.e., supports more users, and guarantees a response time threshold.

A distinguishing characteristic in GeoFeed is that it builds its decision model for each single query q_i instead of the whole system or the set of queries $\mathcal{Q}_u$ for a given user. This means that for a certain user u that has two friends f_i and f_j, GeoFeed may opt to retrieve the messages from f_i through the *spatial push* approach while retrieving the messages from f_j through the *spatial pull* approach. Similarly, for a certain user f that feeds two users u_i and u_j, GeoFeed may opt to have u_i retrieve her messages from f with the *spatial push* approach while u_j retrieve her messages from f with the *spatial pull* approach. Extensive experimental results, based on real and synthetic data, show that (a) GeoFeed is favorable over existing news feed systems, and (b) the accuracy of the GeoFeed decision model in guaranteeing user response time while minimizing the total system overhead.

The closest work to ours is the feeding frenzy approach [23], which is a news feed system equipped with *pull* and *push* approaches to retrieve the most recent news feed items. Unfortunately, the feeding frenzy system cannot be directly applied to the location-aware news feed problem as it does not consider the message location aspect at any of its stages. The only way to turn feeding frenzy to be location-aware is to attach a wrapper around it in a form of a spatial filter, which is extremely inefficient as the spatial filter is applied afterthought. Our proposed location-aware news feed system, GeoFeed, distinguishes itself from feeding frenzy [23] and other systems in having all the following aspects: (1) GeoFeed is designed while having the location-awareness in mind, and thus makes use of the spatial extents of each message to early prune non-relevant messages, (2) GeoFeed modifies the traditional *pull* and *push* approaches to support spatial filters, (3) GeoFeed goes beyond the traditional *pull* and *push* approaches, and introduces the *shared push* approach as a third alternative that reduces the system overhead while not sacrificing the user response time, (4) GeoFeed builds its decision model to minimize the system overhead to support more users, while guaranteeing a certain user response time threshold for each user, (5) GeoFeed gives the message issuer the ability to determine the spatial validity of the each posted message, e.g., the weather service provider may decide a tornado warning is relevant only to followers residing in a certain area.

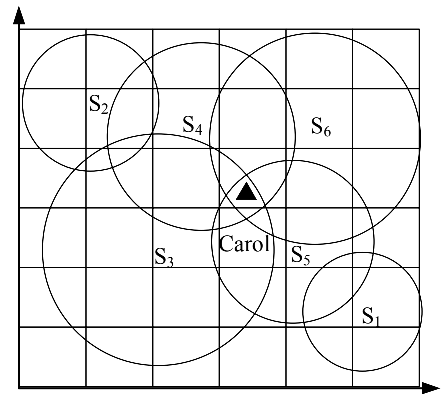
The rest of the chapter is organized as follows: Sections 3.2 gives an overview of GeoFeed. Section 3.3 discusses the *spatial pull*, *spatial push*, and *shared push* approaches. The cost and decision models of GeoFeed are given in Sections 3.4 and 3.5. Section 3.6 discusses GeoFeed with mobile users. Experimental results are given in Section 3.7.

Message	*Timestamp*	*Spatial*	*Content*
M_5	14:30	S_5	Back to hotel.
M_3	14:10	S_3	A nice bar.
M_2	14:04	S_2	Eating at a bar.

(a) Location-based messages posted by user Alice

Message	*Timestamp*	*Spatial*	*Content*
M_6	15:00	S_6	Having coffee.
M_4	14:21	S_4	An accident.
M_1	11:40	S_1	Work finished.

(b) Location-based messages posted by user Bob



(c) Spatial areas of the location-based messages

Figure 3.1: Location-based messages and news feed.

3.2 GeoFeed System Overview

This section gives an overview of GeoFeed as follows:

Location-based messages. GeoFeed users can either send or receive location-based messages to or from their friends with a spatial extent. A location-based message is represented by the tuple: (*MessageID*, *Content*, *Timestamp*, *Spatial*), where *MessageID* and *Content* represent the message identifier and contents, respectively, *Timestamp* is the message generation time, and *Spatial* indicates the spatial extent of the message. Figures 3.1a and 3.1b give examples of six messages, as messages M_2, M_3, and M_5 issued by user Alice and M_1, M_4, and M_6 issued by user Bob . The spatial extents of the six messages are represented by the areas S_1 to S_6 in Figure 3.1c.

System users. A user u, located in $u.location$ with range of interest $u.range$ has a list of friends $\mathcal{F}_u$, who are also considered to be system users, can log on to GeoFeed, and: (a) read all the messages posted from any friend $f_i \in \mathcal{F}_u$ where $u.range$ overlaps with the message spatial extents, and/or (b) post a new message M that should be broadcast to all friends in $\mathcal{F}_u$ that have range of interest overlap with the spatial extents of M. Each user u has a response time threshold $\mathcal{T}_u$ where GeoFeed guarantees to provide the location-aware news feed for u within $\mathcal{T}_u$. The value of $\mathcal{T}_u$ for each user could be set by the system as either a default value for all users, or a value that reflects how valued and appreciated is the user.

Location-based news feed queries. GeoFeed abstracts the location-aware news feed functionality for a user u to a set $\mathcal{Q}_u$ of location-based queries posed to the user's friends $\mathcal{F}_u$. Each query $q_i \in \mathcal{Q}_u$ is posed to a friend $f_i \in \mathcal{F}_u$ to retrieve the k most recent spatially relevant messages to u's location. Then, u may opt to get all the received messages, or select only the k most recent and spatially relevant messages from *all* users. Similar to the feeding frenzy system [23], we will only focus on retrieving k message from *each* user, as an additional filter will be a trivial step. As an example, consider user Carol, depicted by a triangle in Figure 3.1c, which is a friend of Alice and Bob. As the spatial range of interest of Carol includes only her location, she issues two location-based point queries, with $k = 2$, one for Alice (returns M_3 and M_5) and one for Bob (returns M_4 and M_6). Carol can add an additional filter to get the most recent two messages, i.e., M_5 and M_6. The location-based news feed query can also use spatial range instead of the point location to retrieve the most recent k messages that overlap with the querying range.

Query evaluation methods. GeoFeed is equipped with three different approaches for evaluating each query $q_i \in \mathcal{Q}_u$, namely, *spatial pull*, *spatial push*, and *shared push* approaches. Details of these approaches will be discussed in Section 3.3.

Problem formulation. The decision model of GeoFeed can be formulated as follows: *For each location-based news feed query q posed by a user u, find out the best approach among spatial pull, spatial push, and shared push approaches, to evaluate q once u logs on to the system in a future time, such that: (a) the GeoFeed computational overhead for all system queries is minimized, and (b) the response time that u will encounter to get all the requested location-aware news feed is within the required threshold $\mathcal{T}_u$.*

3.3 GeoFeed Query Evaluation

In this section, we present three different query evaluation approaches, *spatial pull*, *spatial push*, and *shared push*, to evaluate a location-based news feed query posed from user u, located at $u.location$, to a friend $f_i \in \mathcal{F}_u$, where $\mathcal{F}_u$ is the list of u's friends. GeoFeed employs these three approaches, monitors their cost (Section 3.4), and then uses its decision model (Section 3.5), to decide on which approach should be used for which queries. The section starts by discussing the underlying data structure, then it goes on to the details of each approach. For simplicity, and without loss of generality, we use location-based point queries where the user range of interest is limited to the user location rather than a spatial area around the user location.

3.3.1 Data Structure

To support its three query evaluation approaches, GeoFeed maintains typical information for each user u that includes ID, name, and location. In addition, GeoFeed maintains the following two data structures for each user u.

List of friends. Each user u maintains a list of friends $\mathcal{F}_u$, where each friend $f_i \in \mathcal{F}_u$ is just another system user that has ID, name, location/querying spatial ranges, list of friends, and gird structure.

Grid index structure. Each user u maintains a grid index structure $\mathcal{G}_u$ that consists of $n \times n$ equal area grid cells, as it is efficient with frequent message updates and works seamlessly for both messages and friends' querying spatial range. Each grid cell $C \in \mathcal{G}_u$ maintains: (1) The set of IDs and spatial ranges of u's friends, whose querying ranges overlap with C, and (2) The set of message IDs produced from u with spatial extents that overlap C. If a message overlaps with several grid cells, its ID will be stored in *every* grid cell it overlaps.

3.3.2 Spatial Pull Approach in GeoFeed

The *spatial pull* approach takes advantage of the grid index structure $\mathcal{G}_{f_i}$ maintained at friend f_i for early spatial pruning at the friend's side. This is the main distinguishing difference between the *spatial pull* approach employed in GeoFeed and the traditional *pull* approach employed in feeding frenzy [23]. With the early pruning pushed inside the *spatial pull* approach, GeoFeed avoids retrieving unnecessary messages as will be encountered in the traditional *pull* approach.

Figure 3.2 gives an example of the main idea of the *spatial pull* approach where user *Alice* needs to get her k spatially relevant messages from her friend *Bob*. The execution flow goes as follows: (1) *Alice* submits her location and the news feed query to *Bob*. (2) *Bob* exploits his grid index structure $\mathcal{G}_{Bob}$ to find out cell C that includes *Alice*'s location, and retrieves all the messages stored in C. (3) *Bob* applies a spatial filter over all the messages returned from $\mathcal{G}_{Bob}$ to only report those messages that include *Alice*'s location (depicted by a black triangle), as there could be messages in C that do not overlap *Alice* location. (4) If more than k messages are returned from the spatial filter, *Bob* forwards the k most recent ones to *Alice* as part of her news feed. *Alice* will get the rest news feed from the other friends.

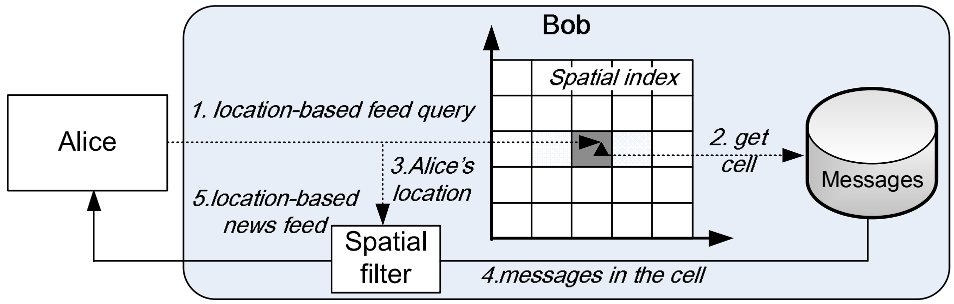


Figure 3.2: Spatial pull approaches in GeoFeed.

3.3.3 Spatial Push Approach in GeoFeed

The *spatial push* approach in GeoFeed pre-computes and stores the answer of the location-based news feed query in a materialized view maintained by the friend f_i. Then, once the user u logs on, u only retrieves the news feed from the materialized view. Although the *spatial push* approach is very appealing to the user, it poses a large overhead over the system resources to continuously maintain the materialized view while the user is offline. Same as in the case of the *spatial pull* approach, the *spatial push* approach is distinct from the traditional *push* approach, used in the feeding frenzy system [23], in that it exploits the grid index structure $\mathcal{G}_{f_i}$ maintained at friend f_i to significantly reduce the overhead of pushing irrelevant messages to the materialized view.

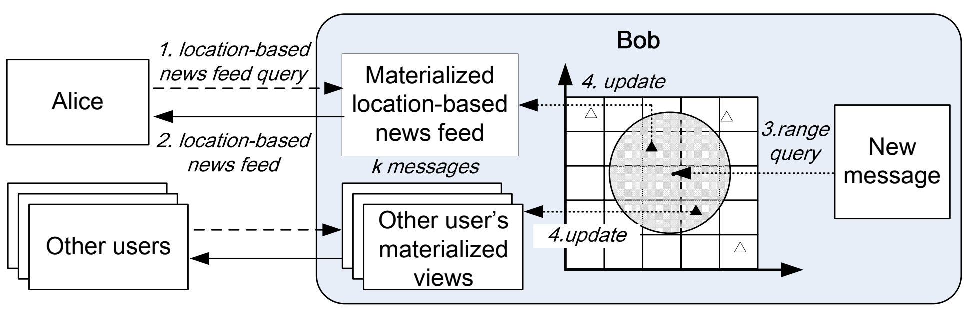


Figure 3.3: Spatial push approaches in GeoFeed.

Figure 3.3 gives an example of the main idea of the *spatial push* approach in GeoFeed where user *Alice* needs to get her k spatially relevant messages from *Bob*. The *spatial push* approach consists of two orthogonal parts, namely, *query processing* and *view maintenance*, detailed as follows: (1) *Query processing. Alice* registers her location with the grid index structure maintained by *Bob*, $\mathcal{G}_{Bob}$. In turn, *Bob* deals with *Alice* location as a continuous location-based point query [74, 75, 76] that needs to be maintained, in a materialized view, even if *Alice* goes offline. Once *Alice* logs on to GeoFeed, she just probes her materialized view, maintained by *Bob*, to retrieve her news feed. (2) *View maintenance.* The friend *Bob* uses his grid index structure, $\mathcal{G}_{Bob}$, to maintain a materialized view for each user employing the *spatial push* approach to retrieve his/her messages from him. Any new message M produced from *Bob* generates a range query over the grid index, $\mathcal{G}_{Bob}$, to retrieve the locations of all friends of *Bob* whose querying ranges overlap the spatial range of M and use the *spatial push* approach to retrieve their news feed from *Bob*. For each of these friends, M is forwarded to the designated materialized view. Figure 3.3 shows *Alice*'s location in $\mathcal{G}_{Bob}$ as a black triangle. The message M produced from *Bob* is depicted as a shaded circle over $\mathcal{G}_{Bob}$, which updates the materialized view of *Alice* among other views of those users who retrieve their news feed from *Bob* with *spatial push*.

3.3.4 Shared Push Approach in GeoFeed

The *shared push* approach in GeoFeed is designed to take advantage of the users locality to reduce the system overhead of the *spatial push* approach while only slightly increasing the response time of location-based news feed queries. The main idea of the *shared push* approach is to maintain one materialized view shared among different spatially co-located users within one grid cell. The benefit is that GeoFeed will maintain much less views than the *spatial push* approach. On the negative side, the messages returned by the shared view need an additional filter to split the answer among the shared views, imposing little overhead over the *spatial push* approach.

Figure 3.4 gives an example of how *Alice* gets her k relevant messages from her friend *Bob*, with the *shared push* approach. The scenario is very similar to the *spatial push* approach with the following two differences in query processing and view maintenance: (1) *Query processing.* As several users share the same materialized view with *Alice*, *Bob* needs to add an additional filter to filter out those news items that are not relevant to *Alice*. (2) *View maintenance.* Instead of maintaining one materialized view for each friend, *Bob* maintains one shared materialized

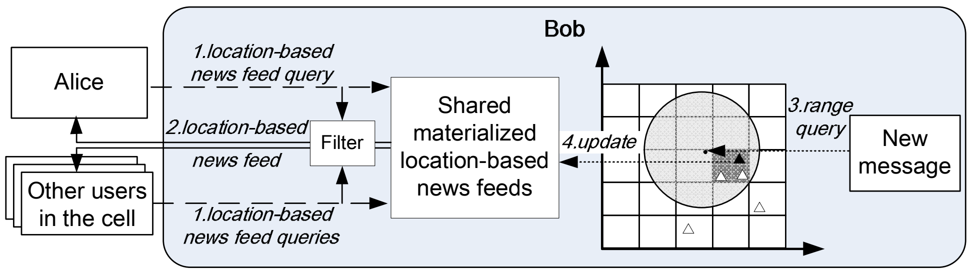


Figure 3.4: Spatial shared push approaches in GeoFeed.

view for all friends with the *spatial push* approach in each cell. Thus, a new message coming out from *Bob* will be inserted in much less materialized views.

The choice of which users to share views together will be discussed in Section 3.5 as part of the GeoFeed decision model.

3.4 GeoFeed Cost Model

This section builds cost models for the *spatial pull*, *spatial push*, and *shared push* approaches in GeoFeed. The cost models are designed to measure: (a) The *system overhead* encountered by GeoFeed to process a location-based news feed query from a user u to a friend f_i, and (b) The *user response time* taken by GeoFeed to prepare the location-aware news feed for a user u from a friend f_i. These cost models will be used in the GeoFeed decision model (Section 3.5) to decide on what would be the best approach to evaluate each news feed query in order to: (a) minimize the GeoFeed system overhead, i.e., providing the ability to scale up GeoFeed to support more users, and (b) ensure that each user u will obtain the news feed within a threshold time $\mathcal{T}_u$. It is important to note that both the GeoFeed cost and decision models for any query issued by user u are designed when u is offline. Once u becomes an online user, GeoFeed sends the initial news feed to u based on the selected approach. As u remains online, new relevant messages to u are just pushed to u as in the *spatial push* approach. So, we ignore any maintenance cost for online users as it will be the same for all approaches. We will first discuss the required data structures to monitor the cost models. Then, we present the cost model.

3.4.1 Data Structure

In order to monitor the cost models, GeoFeed maintains a new data structure, termed *Stat Table*, as follows:

Stat Table. This is a global statistics table that includes the following four statistics for each user u: (1) *Total number of messages* (N_u). This is the total number of messages produced from u since joining the GeoFeed system; updated with every message posted from u. (2) *Response time requirement* ($\mathcal{T}_u$). This is the time threshold set for each user as a hard requirement for GeoFeed to produce the location-aware news feed for u within $\mathcal{T}_u$. The value of $\mathcal{T}_u$ could be set by the system as either a default value for all users, or a value that reflects how valued and appreciated is the user. (3) *Update frequency* ($\mathcal{UF}_u$). This is the average number of messages produced from u per hour; can be either computed from the time that u has joined GeoFeed, over the last hour, or as a weighted average over a certain time window. (4) *Predicated offline time* ($\mathcal{OT}_u$). This is set once u logs off the system, and it reflects the *predicted* offline time (in hours) before u logs on again to the system. $\mathcal{OT}_u$ can be either computed as an average offline time based on user history or similar to last observed offline time.

In addition, GeoFeed adds additional fields to the already maintained data structures, *list of fiends* and *grid structure*:

List of friends ($\mathcal{F}_u$). For each friend $f_i \in \mathcal{F}_u$, we maintain the total number of messages produced from u to f_i, $N_{u \to f_i}$, since u has joined GeoFeed.

Grid index structure ($\mathcal{G}_u$). With each grid cell $C \in \mathcal{G}_u$, we maintain the number of messages $N_{u \to C}$ that are produced from user u and overlap with cell C since u, has joined GeoFeed. This value is updated with each message produced from u.

3.4.2 The Spatial Pull Approach

The cost model for the *spatial pull* approach is developed for each news feed query issued from user u to her friends.

System overhead. As the *spatial pull* approach simply executes a query from u to exploit a grid index structure $\mathcal{G}_{f_i}$, we assume that the system overhead for the *spatial pull* approach is a constant cost, $Cost_{Pull}$, regardless of the user u and the friend f_i. This is mainly because every *spatial pull* query in GeoFeed will go through the same procedure, and hence they all have the same cost.

User response time. As everything in the *spatial pull* approach is done only when the user logs on to the system, the user response time will be the same as the system overhead to evaluate a location-based news feed query, which is $Cost_{Pull}$.

3.4.3 The Spatial Push Approach

Similar to the case of the *spatial pull* approach, the cost model for the *spatial push* approach is developed for each news feed query issued from user u to a friend $f_i \in \mathcal{F}_u$. Since, as we will see, this cost model depends on the number of messages produced from f_i while u is offline, the cost of the *spatial push* approach needs to be reevaluated every time u logs off the system. This is mainly because the GeoFeed decision model is all about what would be the best approach to evaluate u's query the next time u will log on to the system. Based on the decision, GeoFeed will decide whether to start maintaining a materialized view for u at f_i till u logs on the next time (i.e., use the *spatial push* approach), or simply do nothing for u till the next log on time (i.e., use the *spatial pull* approach).

System overhead. The total cost encountered by GeoFeed to support the *spatial push* approach, $Cost_{Push}$, is the summation of two parts: (1) $Cost_{Query}$: The cost to answer the user query issued from u to f_i, and (2) $Cost_{View}$: The cost of maintaining the materialized view at f_i to serve u's query, while u is offline. The details of these two costs are as follows:

1. $Cost_{Query}$. This cost only includes retrieving the news items from the maintained materialized view. We denote this part of the cost as $Cost_{RView}$, which is a constant value as it is a simple selection operation, regardless of u and f_i.
2. $Cost_{View}$. This is the cost of the background processing, done by GeoFeed, to maintain a materialized view for the messages coming from f_i and including u's location. $Cost_{View}$ can be computed as the multiplication of the following two terms: (a) the cost of inserting a message posted by f_i in the materialized view for u, which is a constant cost, $Cost_{IView}$, regardless of f_i and u, and (b) the number of such messages posted from f_i while u is offline. This can be computed as $\frac{N_{f_i \to u}}{N_{f_i}} \times \mathcal{UF}_{f_i} \times \mathcal{OT}_u$, where $\frac{N_{f_i \to u}}{N_{f_i}}$ represents the ratio of messages produced from f_i to u to the total number of messages produced from f_i, $\mathcal{UF}_{f_i}$ is the frequency of messages coming from f_i (the number of messages per hour), and $\mathcal{OT}_u$ is the predicted offline time for the user u (in hours). Notice that $\mathcal{UF}_{f_i}$, N_{f_i}, and $\mathcal{OT}_u$ are stored in the *stat table* while $N_{f_i \to u}$ is stored in the

friend list of f_i.

Then, the cost of the *spatial push* approach, $Cost_{Push}$, is:

$$Cost_{Push} = Cost_{RView} + Cost_{IView} \times \frac{N_{f_i \rightarrow u}}{N_{f_i}} \times \mathcal{UF}_{f_i} \times \mathcal{OT}_u$$

User response time. Once u logs on to GeoFeed to retrieve the news feed from f_i through the *spatial push* approach, GeoFeed simply returns the already maintained materialized view, which has the same cost as the first part of the system overhead, $Cost_{RView}$.

3.4.4 The Shared Push Approach

Unlike the cost models for the *spatial pull* and *spatial push* approaches that were developed for a particular query issued from user u to a friend f_i, and reevaluated every time u logs off GeoFeed, the cost model for the *shared push* approach is: (a) developed for a set of queries $\mathcal{Q_C}$ posed from different users in the same grid cell C to the same friend f_i, and (b) as the queries that share the view are from multiple users, the *shared push* approach is reevaluated every time any user of this shared view logs off.

System overhead. Similar to the *spatial push* approach, the total cost of the *shared push* approach, $Cost_{Shared}$, is the summation of two main parts: (1) $Cost_{SQuery}$: The cost to answer the user queries using the shared view, and (2) $Cost_{SView}$: The cost of maintaining the shared materialized view at f_i. The details of these two costs are as follows:

1. $Cost_{SQuery}$. This is similar to the case of the *spatial push* approach with two differences: (1) The cost for retrieving the query answer from the shared materialized view, denoted as $Cost_{RSView}$, which is slightly higher than the *spatial push* approach, due to an additional filter over the results returned from the shared materialized view. (2) As the cost of the *shared push* approach is computed for a set of queries $\mathcal{Q_C}$, the query cost needs to be multiplied by the number of queries that share the view $|\mathcal{Q_C}|$. Thus, $Cost_{SQuery} = |\mathcal{Q_C}| \times Cost_{RSView}$.
2. $Cost_{SView}$. In a very similar way to the case of the *spatial push* approach, this cost can be computed as: $Cost_{ISView} \times \frac{N_{f_i \rightarrow C}}{N_{f_i}} \times \frac{MBR(\mathcal{Q_C})}{Area(C)} \times \mathcal{UF}_{f_i} \times \min_{\forall u \in Q_C}(\mathcal{OT}_u)$. This includes the following three differences from the case of the *spatial push* approach: (1) We use $Cost_{ISView}$ instead of $Cost_{IView}$ to reflect the cost to update one message

to the shared view. (2) We use $\frac{N_{f_i \to C}}{N_{f_i}} \times \frac{MBR(\mathcal{Q_C})}{Area(C)}$ instead of $\frac{N_{f_i \to u}}{N_{f_i}}$ as the ratio of the messages produced from f_i in cell C to the total number of massages produced from f_i, multiplied by the ratio of the covered area of the minimum bounding rectangle (MBR) of the point queries sharing the same materialized view to the area of the grid cell C. (3) We use $\min_{\forall u \in Q_C}(\mathcal{OT}_u)$ instead of $\mathcal{OT}_u$ to reflect the time when the first user of the shared view in cell C logs off the system, in which the cost model needs to be reevaluated.

Then, the cost of the *shared push* approach, $Cost_{Shared}$, is:

$$
\begin{aligned}
Cost_{Shared} = |\mathcal{Q_C}| \times Cost_{RSView} + Cost_{ISView} \\
\times \frac{N_{f_i \to C}}{N_{f_i}} \times \frac{MBR(\mathcal{Q_C})}{Area(C)} \times \mathcal{UF}_{f_i} \times \min_{\forall u \in Q_C}(\mathcal{OT}_u)
\end{aligned}
$$

User response time. Similar to the case of the *spatial push* approach, the cost here will be to only read from the shared materialized view, which is $Cost_{RSView}$.

3.5 GeoFeed Decision Model

In this section, we discuss the GeoFeed decision model that is applied for each query posed by a user u and is triggered every time u logs off from GeoFeed. The goal of this decision model is to find out the best approach, among the *spatial pull*, *spatial push*, and *shared push* approaches, to evaluate each news feed query issued from user u to a friend $f_i \in \mathcal{F}_u$, for the next time that u will log on to the system, i.e., after $\mathcal{OT}_u$ time units. The objective is to minimize the system overhead encountered by GeoFeed while ensuring that user u will get all the requested news feed within the time threshold $\mathcal{T}_u$. Once u logs on to the system, GeoFeed issues $|\mathcal{F}_u|$ news feed queries as one for each friend $f_i \in \mathcal{F}_u$. Each query will be evaluated using the approach that was selected by the GeoFeed decision model. As u remains online, GeoFeed keeps pushing new messages from any of u's friends to u as news feed. Once u logs off the system, the decision model will be reevaluated again based on the new expected $\mathcal{OT}_u$.

The GeoFeed decision model consists of three main steps, detailed in the rest of this section: (1) Step 1, *Response time guarantee*, finds out the maximum number of *spatial pull* queries, $NQPull_u$, that user u can afford while having the news feed within $\mathcal{T}_u$, (2) Step 2, *Pull vs. Push selection*, decides on what are these $NQPull_u$, out of all the queries posed by u that will be assigned to the *spatial pull* approach, other queries will be assigned to the *spatial push* approach, and (3) Step 3, *Refinement with shared push*, finds out if using the *shared push* approach for any grid cell C maintained at friend f_i can reduce GeoFeed system overhead.

3.5.1 Step 1: Response Time Guarantee

Objective. Since the *spatial pull* queries are mostly favorable to the system due to their lower overhead, yet, they result in high response time, this step finds out the number $NQPull_u$ as the maximum number of *spatial pull* queries that u can afford to guarantee that the news feed will be delivered to u within time threshold $\mathcal{T}_u$.

Main idea. In terms of *user response time*, the *spatial push* approach gives much lower response time than that of the *spatial pull* approach, i.e., $Cost_{RView} \ll Cost_{Pull}$, regardless of the user u and the friend f_i. In the mean time, the user may not actually feel the difference between the *spatial push* and the *shared push* approaches as their user response times are very close to each other, though the latter is slightly higher. Thus, from the user perspective, the user would always like to avoid using the *spatial pull* approach and have all the queries evaluated with either the *spatial push* or *shared push* approaches. However, a large number of views introduces significant system overhead to maintain them. To balance between these two contradicting factors, GeoFeed uses the response time requirement $\mathcal{T}_u$ for each user u to decide that u can tolerate having some queries evaluated with the *spatial pull* approach while the overall response time for all queries is less than $\mathcal{T}_u$.

Algorithm. Since user u has to issue $|\mathcal{F}_u|$ location-based news feed queries, we want to find the maximum number of queries $NQPull_u$ among $|\mathcal{F}_u|$ that can be evaluated with the *spatial pull* approach while ensuring that u will get the required news feed within the time threshold $\mathcal{T}_u$. If $NQPull_u$ queries will be evaluated using the *spatial pull* approach with response time $Cost_{Pull}$, then the rest of queries posed by user u, i.e., $|\mathcal{F}_u| - NQPull_u$, will be evaluated through either the *spatial push* or *shared push* approach. Given that these two latter costs are very close, with $Cost_{RSView}$ is slightly higher, then the following inequality should hold for any user u:

$$NQPull_u \times Cost_{Pull} + (|\mathcal{F}_u| - NQPull_u) \times Cost_{RSView} < \mathcal{T}_u$$

Thus, we can get $NQPull_u$ using the following equation:

$$NQPull_u = \left\lfloor \frac{\mathcal{T}_u - |\mathcal{F}_u| \times Cost_{RSView}}{Cost_{Pull} - Cost_{RSView}} \right\rfloor$$

3.5.2 Step 2: Pull vs. Push Selection

Objective. Now that GeoFeed finds out that u can afford having $NQPull_u$ queries with the *spatial pull* approach, it is the objective of this step to decide which $NQPull_u$ queries out of the total of $|\mathcal{F}_u|$ queries that will be evaluated using the *spatial pull* approach. The decision is taken to minimize GeoFeed system overhead.

Main idea. The main idea of this step is to employ the following two concepts: (1) Two queries from the same user u to friends f_i and f_j may have different system overhead costs $f_i.Push$ and $f_j.Push$ for the *spatial push* approach based on the update frequency and spatial distribution of messages coming from f_i and f_j. Assuming that $f_i.Push > f_j.Push$, and that u can afford having only one query with the *spatial pull* approach, then, GeoFeed will select the *spatial push* approach for f_j, as it has lower system overhead, leaving f_i to be evaluated with *spatial pull*. Thus, if u would accept having $NQPull_u$ queries with the *spatial pull* approach, then the main idea here is to select the set of $NQPull_u$ queries that are the worst in terms of system overhead in *spatial push* to be assigned to the *spatial pull* approach. All other queries will be assigned to the *spatial push* approach. The main reason here is that from the user perspective, it does not really matter which queries will be selected as *spatial pull*, while this decision has significant impact on the system overhead. (2) In some cases, using the *spatial push* approach for a certain query may have less system overhead than using the *spatial pull* approach. Consider, for example, a user u with an expected very short offline time $\mathcal{OT}_u$. In this case it is better for GeoFeed to incrementally maintain a view of the last reported answer for u for a short time rather than reevaluating u's query with the *spatial pull* approach. This will be also favorable to the user for the much lower user response time. Thus, it may be desirable to have less than $NQPull_u$ queries using the *spatial pull* approach, if there are queries among the worst $NQPull_u$ that still have lower system overhead when using the *spatial push* approach.

Algorithm. Algorithm 1 gives the pseudo code for Step 2 in the GeoFeed decision model. The input to the algorithm is the number of queries $NQPull_u$ that u can afford having in the *spatial pull* approach, along with the data structure used to compute the cost model. The algorithm starts by calculating the system overhead cost for using the *spatial push* approach for each query posed by user u to a friend f_i. While calculating the cost, we insert u's friends

Algorithm 1 Step 2: Pull vs. Push Selection

Input: (1) $NQPull_u$; the number of queries with the *spatial pull* approach that u can afford, and (2) The data structure used for computing the cost model.

Output: Set the decision $f_i.Decision$ for each query posed from u to a friend $f_i \in \mathcal{F}_u$ to either *spatial pull* or *spatial push*.

for each friend $f_i \in \mathcal{F}_u$ **do**
    $f_i.Push \leftarrow Cost_{RView} + Cost_{IView} \times \frac{N_{f_i \rightarrow u}}{N_{f_i}} \times \mathcal{UF}_{f_i} \times \mathcal{OT}_u$
    Insert f_i in a Max Heap $\mathcal{MH}$ based on $f_i.Push$
end for
PullCount $\leftarrow 0$
$f_m \leftarrow$ The top element from the heap $\mathcal{MH}$
while *PullCount* $< NQPull_u$ AND $f_m.Push > Cost_{Pull}$ **do**
    Remove f_m from the Max Heap $\mathcal{MH}$
    $f_m.Decision \leftarrow$ ***spatial pull***
    $f_m \leftarrow$ The top element from the heap $\mathcal{MH}$
    PullCount $\leftarrow$ *PullCount* + 1
end while
for each remaining f_h in the heap $\mathcal{MH}$ **do**
    $f_h.Decision \leftarrow$ ***spatial push***
end for

in a maximum heap structure ordered by the calculated cost. Then, we keep removing friends from the maximum heap one by one, and assign them to the *spatial pull* approach, till any one of these two stopping conditions takes place: (a) the number of queries with the *spatial pull* approach has reached its maximum, which is $NQPull_u$. In this case, assigning more *spatial pull* queries will increase the user response time for u to be more than the threshold $\mathcal{T}_u$, or (b) we find a query that has lower system overhead cost with the *spatial push* approach than the cost of the *spatial pull* approach. In this case, it is better for both the system and the user to have this query, and all subsequent queries, with the *spatial push* approach. Note that all subsequent queries will have a lower *spatial push* overhead cost than the current query, and hence will also favor the *spatial push* approach. Once any of these two conditions is satisfied, we assign all the remaining queries in the maximum heap data structure to the *spatial push* approach.

3.5.3 Step 3: Refinement with Shared Push

Objective. Up to now, we have an initial decision for each query posed from u that satisfies the requirement $\mathcal{OT}_u$. In this step, we look further for all queries with the *spatial push* approach, and find out if some of them can be grouped together to use the *shared push* approach, and hence amortize the cost of maintaining a materialized view among several queries. As the response times of the *spatial push* and *shared push* approaches are similar, and we have already used the *shared push* cost in Step 1, the refinement step will never break the response time requirement for any user.

Main idea. The main idea here is to go through all the grid cells of each user in the system, and check if the *shared push* approach will be useful. Notice that u is located in $|\mathcal{F}_u|$ grid cells as one per each friend $f_i \in \mathcal{F}_u$. In any of these grid cells, using the *shared push* approach may be clearly favorable if most of the users in this cell are using the *spatial push* approach. In this case, the system overhead to maintain the shared view is amortized over these users. Unfortunately, there is no magic number for the number of *spatial push* queries in a cell that would call for going towards the *shared push* approach, as this depends on several factors that include the first offline time for any user in the cell, the frequency of users' updates, and the size of the minimum bounding rectangle of the shared view. So, a full computation of the gain/loss from using the *shared push* approach for all the users in the cell needs to be done before a decision is taken.

Algorithm. This step can be evaluated by having a loop over all the $|\mathcal{F}_u|$ grid cells that include u. For each such cell C at friend f_i, we will calculate two costs: (1) $Cost_{shared}$, as the cost of using the *shared push* approach for all the users in this cell. This is computed as described in Section 3.4.4, and (2) $Cost_C$, as the current cost encountered for the users in C that currently use the *spatial push* approach. To compute this cost, we will need to go through all the users with the *spatial push* approaches and compute their current cost. At the end if $Cost_{shared} < Cost_C$, we set the decision to apply the *shared push* approach for all users in C with *spatial push* approach, otherwise, we do not change the decisions taken so far.

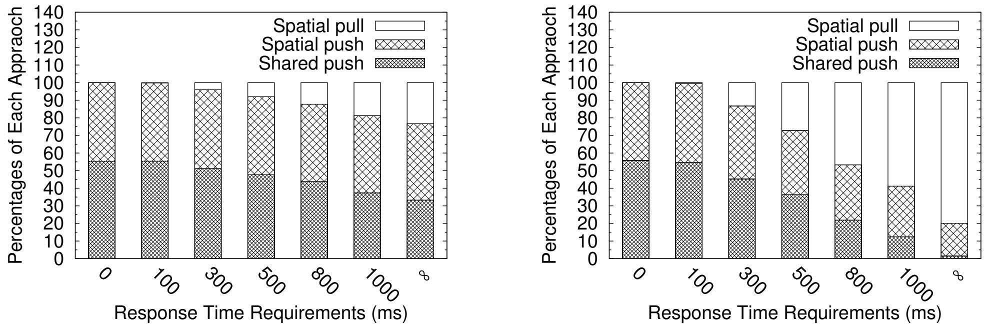
3.6 GeoFeed for Mobile Users

All our previous discussions consider static registered locations for users logging on to GeoFeed. So, once a user logs off GeoFeed, we compute the cost and decision models for that user based on its registered location. User locations can be registered with GeoFeed either explicitly from the user or implicitly by detecting that a user is frequently logging on from a certain location. If a user has multiple registered static locations, e.g., home and work, GeoFeed treats each location separately, where the cost and decision models can be different for each registered location. As mentioned earlier, once a user is logged on, incoming news feed will just be pushed to the user based on the first location he/she logged on from.

In the case of mobile users, i.e., users keep moving during a login session, just pushing the incoming news feed based on the initial logging location is not applicable as users keep changing their locations. On the other hand, pulling the news feed for each new location has a prohibitive cost due to the high frequency of location changes. To this end, we modified the *spatial push* approach to support mobile users. The basic idea is to extend a user u's point location to be the whole grid cell C in which u is located in. Now, C is considered as the region of interest of user u where the news feed will be retrieved as those messages from u's friends and overlap with C. Then, a filter will be added to show only those messages that overlap with u' location. As long as u moves within its cell C, incoming messages that overlap with C are pushed to u as in the *spatial push* approach. Once u moves out of C to another grid cell C', GeoFeed employs the *spatial pull* approach to retrieves news feed overlap with C'.

3.7 Experiments

This section gives experimental evaluation of GeoFeed based on an actual system implementation in PostgreSQL database management system [77]. We compare GeoFeed against an adaptation of the feeding frenzy approach [23], an industrial solution from Yahoo!, through an additional spatial filter as well as variations of the query evaluation approaches within GeoFeed. All experiments are based on a mixture of real and synthetic data. The real part comes from Twitter messages collected by our web crawler via Twitter Search API [78] by continuously issuing a query to the API with a spatial range of 150 $\times$ 150 miles space (approximately the size of the state of Minnesota). We query the API continuously for one week, and got 646,697 geotagged distinct tweets. The Twitter search API returns the location information with each tweet as either a longitude/lattitude coordinate or a semantic location, e.g., a city name, in which we use TinyGeoCoder [79] and Google GeoCoding [80] APIs to map it to longitude/latitude coordinates. Then, we randomly generate message spatial extents as circular areas centered at each message issuing location with a radius generated uniformly from 5 to 20 miles. We use a synthetic data set of 10,000 users randomly distributed over the map of Minnesota, where we randomly associate the real tweets to the synthetic users. Mostly taken from Facebook statistics [81], and unless mentioned otherwise, each user has an average of 150 friends, eight hours



(a) Offline Time = 1 hour.

(b) Offline Time = 8 hours.

Figure 3.5: Inside GeoFeed Decision Model.

offline time before logging on, 500ms response time requirement, range of interest includes the user location only, and generates 90 messages per month. All average values are generated using Zipf distribution [82] (with skewness 0.5) that gives higher probabilities for smaller values. Experiments were evaluated on a server computer with Intel Core 2 Quad CPU 2.83GHz processor and 4 GB RAM with Ubuntu Linux 9.04.

To build the cost model, we ran 1,000 queries using each approach. We find that $Cost_{Pull}$=7.8ms, $Cost_{RView}$=0.5ms, $Cost_{RSView}$=0.8ms, $Cost_{IView}$=21.2ms and $Cost_{ISView}$=22.5ms. This confirms our earlier assumption that $Cost_{RView} \ll Cost_{Pull}$, and $Cost_{RView}$ is relatively close to $Cost_{RSView}$. We use these constant values in our GeoFeed decisions. The rest of this section is as follows: Section 3.7.1 gives an inside look on how GeoFeed takes its decisions. Comparison with other approaches is in Section 3.7.2. Section 3.7.3 tests GeoFeed performance with respect to various parameters by simulating the workload for one day with 300 users.

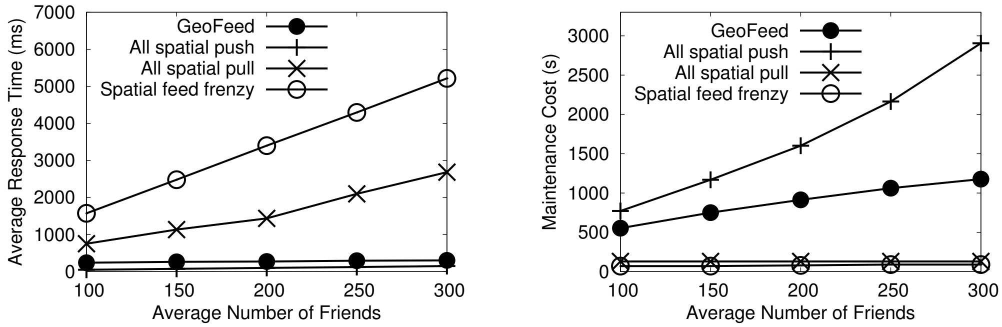
3.7.1 Inside GeoFeed

Figure 3.5 gives an inside view of how GeoFeed smartly takes its decisions in selecting the right query evaluation approach among *spatial pull*, *spatial push*, and *shared push* approaches for each user query issued from a user u to a friend $f_i \in \mathcal{F}_u$. We vary the user response time requirements $\mathcal{T}_u$ from zero to infinity for a user offline time of one hour (Figure 3.5a) and eight hours (Figure 3.5b). For each value of $\mathcal{T}_u$, we plot the ratio of queries evaluated with *spatial pull*, *spatial push*, and *shared push* approaches.

This experiment gives the following four very interesting insights: (1) With the increase of $\mathcal{T}_u$, more *spatial pull* approaches are selected. This is mainly because GeoFeed feels that it has a relaxed $\mathcal{T}_u$, so, it goes for minimizing the system overhead by having more *spatial pull* approaches that does not cost much of system overhead. In the mean time, we have less *shared push* approaches with the increase of $\mathcal{T}_u$, which is natural as we have less *spatial push* approaches, which decreases the probability that GeoFeed finds queries that can be shared together. (2) When $\mathcal{T}_u$=0ms, (i.e., the user needs the news feed as fast as possible), no *spatial pull* approaches are applied, as they definitely pose more query response time for system users. However, it is interesting to notice that not all the queries are evaluated by the *spatial push* approach where a significant portion of the queries use the *shared push* approach. This is mainly due to our valid assumption that the query costs for both the *spatial push* and *shared push* approaches are similar. So, GeoFeed aims to reduce the system cost through having more of the *shared push* approach. (3) When $\mathcal{T}_u$=∞ (i.e., u does not have any response time requirement), GeoFeed aims to only minimize the system overhead through employing much of the *spatial pull* approach. However, it is interesting to notice that some queries are still evaluated with other approaches, especially with smaller offline time (Figure 3.5a). This is mainly because the GeoFeed decision model takes into account the case that the *spatial push* approach may be cheaper than the *spatial pull* approach, which takes place with low update frequency and/or short user offline time. (4) Comparing Figures 3.5a and 3.5b together shows that with a smaller offline time, more *spatial push* approaches are applied. As a smaller offline time means that the user will log soon again to GeoFeed, so, it is better to maintain the materialized view in the *spatial push* approach rather than executing the query from scratch upon the next log on time as in the *spatial pull* approach.

3.7.2 Comparison with Other Approaches

Figure 3.6 compares GeoFeed performance against: (a) an adaptation of *feeding frenzy* [23] to handle spatial data through an additional spatial filter, (b) having all the queries evaluated with the *spatial pull* approach, which is the state-of-the-art solution optimization in location-based systems, that pushes the spatial pruning to the bottom and (c) having all the queries evaluated with the *spatial push* approach, which is the state-of-the-art optimization in the publish-subscribe systems, that prunes out the unnecessary update notices early. We vary the average number of friends for each user from 100 to 300, and measure the average response time (Figure 3.6a) and system overhead cost (Figure 3.6b). It is clear that feeding frenzy has a very bad



(a) Average Response Time.

(b) Total System Overhead.

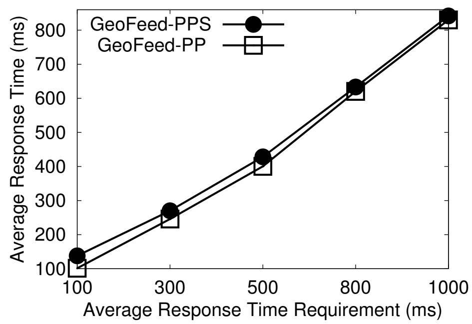
Figure 3.6: Different Friends Numbers.

user response time as the spatial filter is applied afterthought, and it does not take the spatial aspects of the messages in its decisions. Though the *all spatial pull* approach gives better response time than feeding frenzy, but it is still unacceptable as it evaluates the query from scratch for each friend. The performance of GeoFeed and *all spatial push* are very similar in terms of response time, and almost have a steady performance with the increase of the number of friends, which is orders of magnitude better than that of both the feeding frenzy and *all spatial pull* approaches. With respect to system overhead, the *all spatial push* approach has a much higher cost than that of GeoFeed (about double cost for 300 friends). This is mainly due to the large number of materialized views maintained for the *all spatial push* approach.

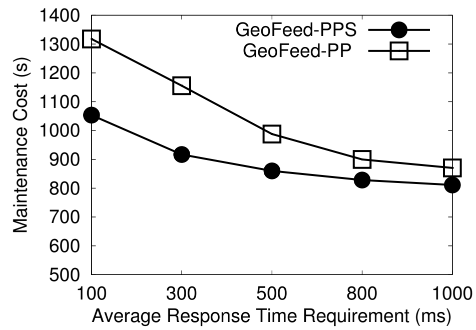
From this experiment, we conclude that both feeding frenzy and *all spatial pull* approaches are completely impractical due to their unacceptable user response time, and hence we will not consider them later. Similarly, we will not consider the *all spatial push* approach later due to its clearly higher overhead cost than GeoFeed.

3.7.3 GeoFeed Performance

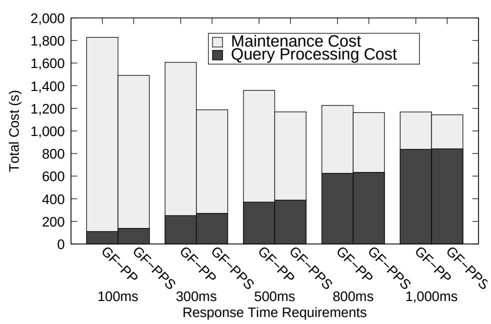
In this section, we compare the performance of two variations of GeoFeed; one with only the *spatial pull* and *spatial push* approaches (termed GF-PP or GeoFeed-PP), and the other one is our complete GeoFeed system (termed GF-PPS or GeoFeed-PPS) with different parameters: (1) user response time requirement $\mathcal{T}_u$, (2) offline time, (3) news update rate, (4) user spatial distribution, (5) grid cell granularity, (6) user querying range, and (7) GeoFeed with mobility. For a fair comparison, we use the same fixed scale in x-axis for the most experiments when



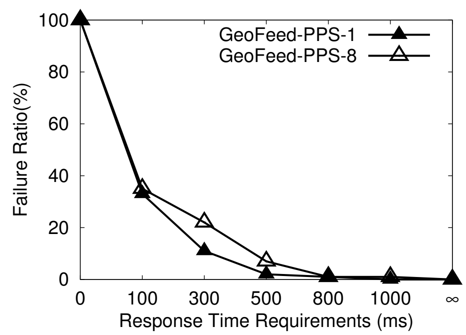
(a) Average Response Time.



(b) Maintenance Cost.



(c) Total Cost.



(d) Fail Ratio.

Figure 3.7: Response time requirements.

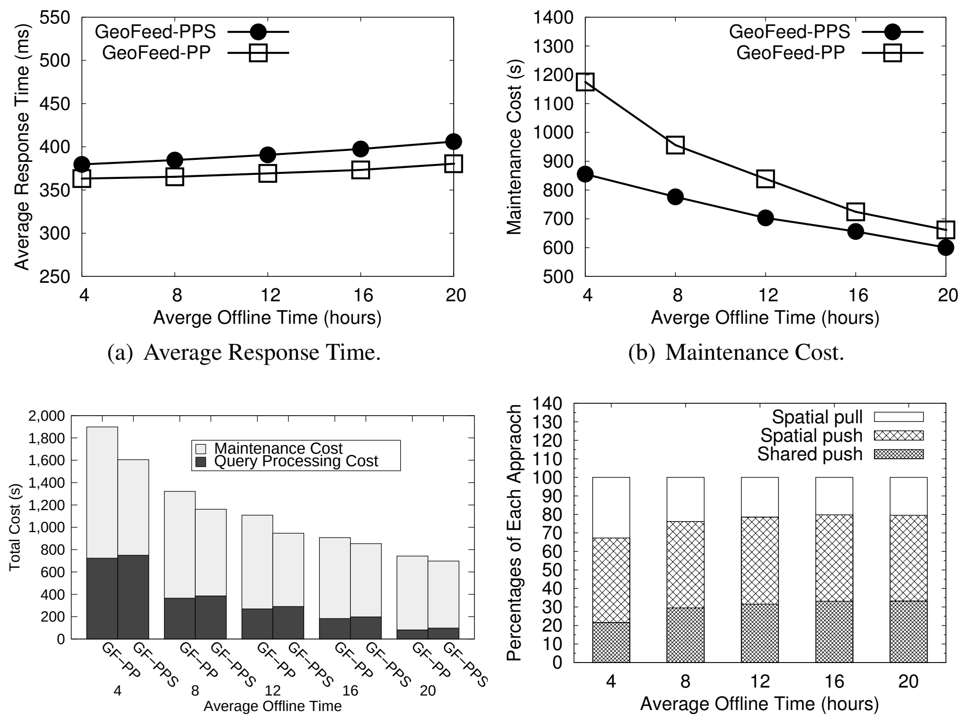
comparing the average response time and total update cost between GF-PP or GeoFeed-PP.

Response Time Requirements. Figure 3.7 gives the impact of user response time requirements on the performance of GeoFeed-PPS and GeoFeed-PP, where the required response time varies from 100 to 1,000 ms. When the response time requirement becomes less strict, the system can use more *spatial pull* approaches to process more location-based queries. Since the response time of the *spatial pull* approach is higher than that of the *spatial push* and *shared push* approaches, the response time of both GeoFeed-PPS and GeoFeed-PP gets worse with longer required response times (Figure 3.7a). When there are more location-based queries selected to use the *spatial pull* approach, i.e., a smaller number of queries is processed by the *spatial push* approach, the chance of grouping queries to use the *shared push* approach becomes slimmer. Thus, the improvement of GeoFeed-PPS on the view maintenance cost of materialized views reduces (Figure 3.7b). From Figure 3.7c, we can observe that with tight requirements, there is significantly large maintenance cost than query processing cost, as GeoFeed tends to avoid using *spatial pull* in order to be able to satisfy the user requirements, and thus has to pay maintenance cost for using *spatial push* and *shared push*. With relaxed requirements (increase of $\mathcal{T}_u$), GeoFeed tends to optimize its maintenance cost in favor of having more query processing cost, such that the overall total cost is decreased. Since maintenance cost optimization reduces the use of *spatial push*, it also reduces the possibility of having *shared push* refinement, and hence the performance of both GeoFeed variations becomes similar. This shows that the use of *shared push* refinement is more clear when the system is overloaded either with tight requirements, short offline times, or high update frequencies.

Figure 3.7d gives the ratio of user queries that exceed the user response requirements for GeoFeed-PPS with average offline time of one hour and eight hours. We vary $\mathcal{T}_u$ from zero to ∞. What is interesting here is the failure ratio is exponentially decreasing with relaxed time constraints. Also, we have less failure with smaller offline time, as more queries use the *spatial push* approach which has more deterministic user response time.

User Offline Time Periods. Figure 3.8 compares the performance of GeoFeed-PPS and GeoFeed-PP with respect to various user offline time periods from 4 to 20 hours. Increasing the user offline time period also increases the cost of the *spatial push* approach because the cost of maintaining materialized views during a user offline time period is higher. As a result, the system selects more queries to use the *spatial pull* approach. Since the response time of the *spatial pull* approach is higher than that of the *spatial push* approach, the response time of both GeoFeed-PPS and GeoFeed-PP gets higher when the user offline time period increases, as depicted in Figure 3.8a. Since the number of location-based queries using the *spatial push* approach decreases as the user offline time period gets longer, smaller numbers of queries can be grouped to use the *shared push* approach; and hence, the improvement of GeoFeed-PPS on the total view maintenance cost is smaller. From Figure 3.8c, we can observe that with the increase of offline time, both the query processing and maintenance cost decrease as less queries are posed to the system, and most of them are evaluated with *spatial pull*. Overall, with the increase of offline time, we have a similar performance of GeoFeed-PPS and GeoFeed-PP, because there are less *spatial push* queries, which is illustrated in Figure 3.8d, and hence the impact of the *shared push* refinement is reduced.

News Update Rates. Figure 3.9 compares the performance between GeoFeed-PPS and GeoFeed-PP by varying the average number of news update rates per month from 30 to 150. Since a higher news update rate results in more messages generated in the system, both GeoFeed-PPS

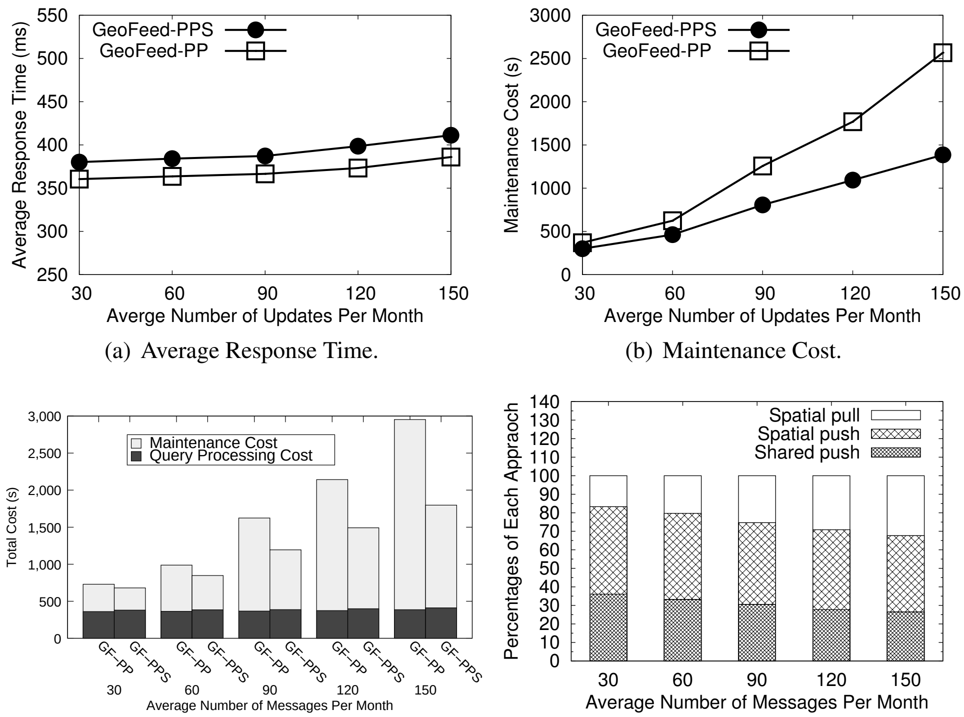


(a) Average Response Time. (b) Maintenance Cost.

(c) Total Cost. (d) Inside GeoFeed Decisions.

Figure 3.8: User offline time periods.

and GeoFeed-PP need to use the grid index to prune more irrelevant messages; and thus, their response time increases (Figure 3.9a). When the news update rate increases, both algorithms have to update materialized views more frequently. Since the materialized view update cost of the *shared push* approach is shared by a group of users, it is more efficient than updating each materialized view of the *spatial push* approach individually. Therefore, GeoFeed-PPS performs much better than GeoFeed-PP in terms of the view maintenance cost of maintaining materialized views when the news update rate gets higher (Figure 3.9b). In Figure 3.9c, the average update frequency is varied from 30 to 150 per month. With the increase of the update frequency, the maintenance cost increases significantly for both approaches due to the need of more updates to the materialized views. However, GeoFeed-PPS has less total cost, as the *shared push* approach reduces the number of insertions to the materialized views. It shows the efficiency gained from sharing multiple views together to significantly reduce the maintenance cost. With respect to query processing cost, there is a slight increase for both approaches with the growth



(a) Average Response Time.

(b) Maintenance Cost.

(c) Total Cost.

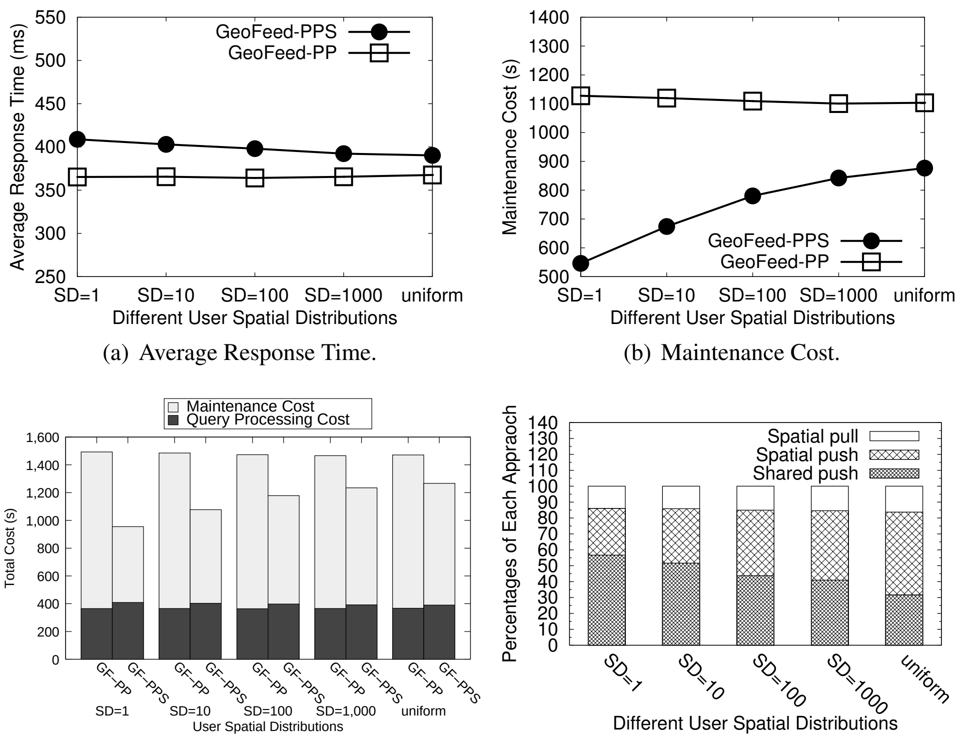
(d) Inside GeoFeed Decisions.

Figure 3.9: News update rates.

of the update frequency, which is illustrated by the decisions inside GeoFeed in Figure 3.9d. Overall, GeoFeed-PPS gives much lower total cost than GeoFeed-PP.

User Spatial Distributions. Figure 3.10 gives the experimental results of GeoFeed-PPS and GeoFeed-PP with respect to two different user distributions: (1) Uniform distribution where users are randomly distributed in the map and (2) Gaussian distribution where users are distributed in the map according to different standard deviations SD that indicate different user densities in a certain range distance of a given location. The Gaussian distribution is mainly used to simulate a hot spot area in the map, i.e., the downtown area in the city, to further explore the impact of user spatial locality and the advantage of applying the *shared push* approach. A smaller value of SD indicates that users are concentrated in a denser area. In this experiment, SD varies from 1 to 1,000.

Since the user density does not affect the selection between the *spatial pull* and *spatial push* approaches, varying the user density only slightly affects the performance of GeoFeed-PP



(a) Average Response Time.

(b) Maintenance Cost.

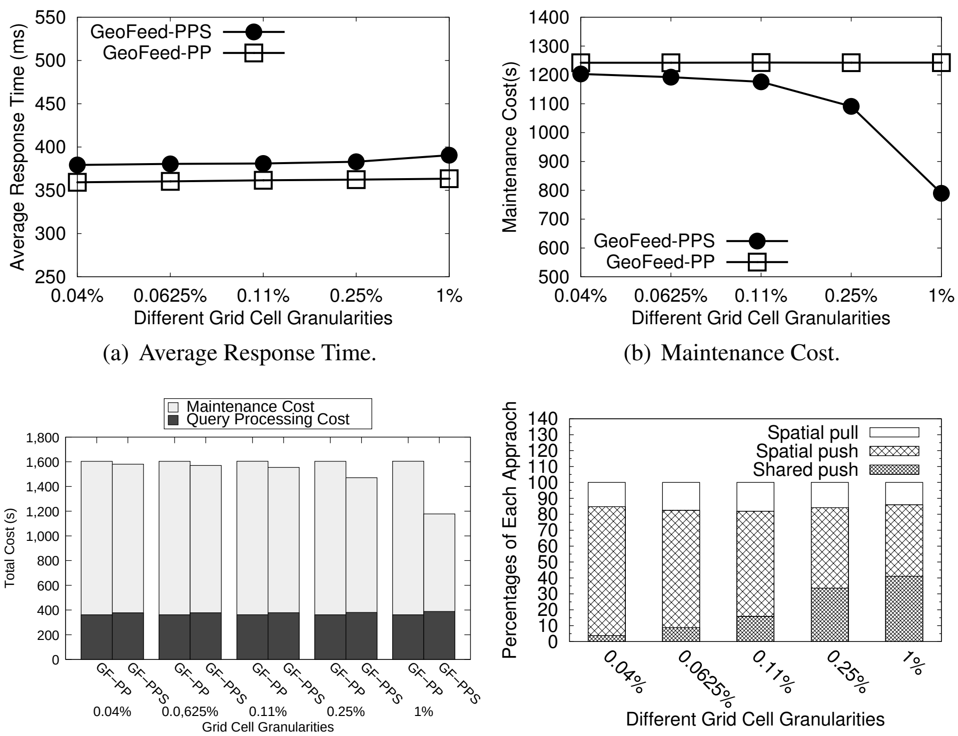
(c) Total Cost.

(d) Inside GeoFeed Decisions.

Figure 3.10: User distributions.

(Figures 3.10a to 3.10c). However, the response time of GeoFeed-PPS increases as the users are distributed more densely. This is because the system with a higher user density has a higher probability to use the *shared push* approach to process user queries, as given in Figure 3.10d, which increases the query response time. On the positive side, using the *shared push* approach to process more user queries leads to a lower update cost of maintaining shared materialized views (Figure 3.10b). As a result, the view maintenance cost of GeoFeed-PPS performs much better than that of GeoFeed-PP when the user distribution becomes denser.

Grid Cell Granularities. Figure 3.11 depicts the performance of GeoFeed-PPS and GeoFeed-PP with respect to various cell granularities of the grid index which is defined as a ratio of the cell area to the total system area. This experiment considers five different grid cell granularities 0.04%, 0.0625%, 0.11%, 0.25% and 1%. The results give that the grid cell size only slightly affects the performance of GeoFeed-PP (Figures 3.11a to 3.11c). On the other hand, when the



(a) Average Response Time.

(b) Maintenance Cost.

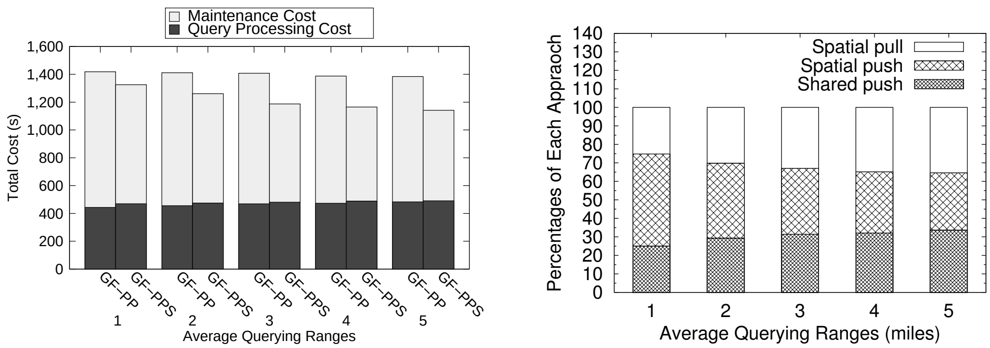
(c) Total Cost.

(d) Inside GeoFeed Decisions.

Figure 3.11: Grid cell granularities.

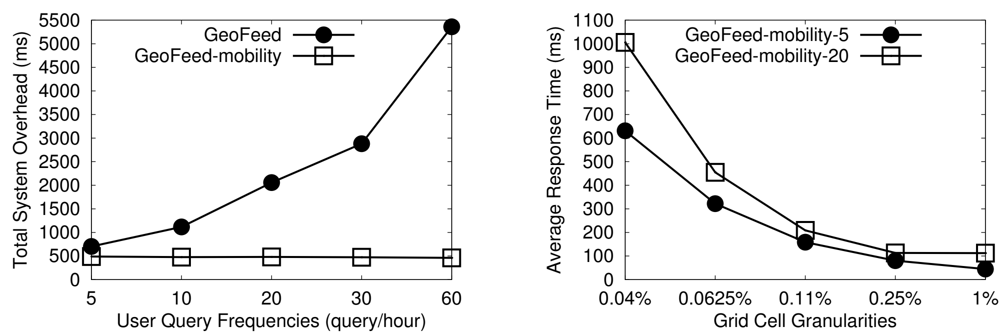
grid cell size gets larger, there is a higher chance for GeoFeed-PPS to use the *shared push* approach to share a materialized view with a group of user queries, as confirmed by Figure 3.11d. Due to the fact that the response time of the *shared push* approach is slightly higher than the *spatial push* approach, the response time of GeoFeed-PPS increases as the grid cell size gets larger.

User Querying Ranges. Figure 3.12a depicts the performance of GeoFeed-PPS and GeoFeed-PP with different user querying ranges with respect to varying from 1 mile to 5 miles. Figure 3.12b demonstrates the decisions inside GeoFeed system. With the increase of the user querying ranges, more queries are executed with *spatial pull* approach, because the larger querying range has more chance to get the message updates, which increases *spatial push* cost to maintain the materialized view. As a result, the total querying process cost in Figure 3.12a increases with the growth of user querying ranges. Moreover, more *shared push* approaches are adapted in the system, because the higher cost of *spatial push* makes *shared push* approach



(a) Total Cost. (b) Inside GeoFeed Decisions.

Figure 3.12: User Querying Ranges.



(a) Query Frequencies. (b) Grid Cell Granularities.

Figure 3.13: GeoFeed with Moving Users.

more efficient. As a result, our GeoFeed-PPS becomes more efficient with a lower overall system overhead than GeoFeed-PP with the increase of user querying ranges.

GeoFeed Mobility. Figure 3.13a depicts the performance of GeoFeed and GeoFeed-Mobility for mobile users with respect to varying their query frequencies from 5 to 60 queries per hour with the user's traveling speed at 5 miles per hour and the grid cell granularity as 0.25%. The results show that GeoFeed-Mobility is much more scalable than the original GeoFeed. In fact, the system overhead of GeoFeed-Mobility is not dependent of the user query frequency because GeoFeed-Mobility only needs to push new messages to a user when the user moves across grid cells.

Figure 3.13b gives the results of GeoFeed-Mobility for mobile users moving at two mobility speeds (i.e., 5 and 20 miles per hour) in different grid cell granularities, increasing from

0.04% to 1%, with the default query frequency at 10 queries per hour. The results show that the response time of GeoFeed-Mobility improves as the grid cell size gets larger because the users take longer time to move outside a grid cell; thus, the users can compute their query answers locally without enlisting the server for help. Similarly, GeoFeed-Mobility with the lower mobility speed provides the better response time, because the users can stay in a grid cell for longer time.

3.8 Summary

We have presented the GeoFeed system; a location-aware news feed system that takes into account the spatial extents of messages and user locations when deciding upon the selected news feed. GeoFeed is equipped with three different approaches, namely *spatial pull*, *spatial push* and *shared push* for delivering the news feed to its users. Based on an accurate developed cost model for each approach, GeoFeed employs a smart decision algorithm that decides about using these approaches in a way that: (a) minimizes the system overhead for delivering the location-aware news feed, and (b) guarantees a certain response time for each user to obtain her location-aware news feed. GeoFeed further extends the *spatial push* approach to support the moving users. Experimental results, based on real and synthetic data, show that GeoFeed is favorable over existing news feed systems, with a minimal system overhead.

Chapter 4

GeoRank: Location-Aware News Feed Ranking

4.1 Introduction

Social networking services, e.g., Twitter and Facebook, and news aggregators, like iGoogle and MyYahoo!, have become one of the most popular web services. One of their main functionalities is the news feeds [23], where users can receive news items posted from their subscribed sources, e.g., friends or news sources of interest. With the widely usage of location information, many news items are now associated with geographical information, e.g., geo-tagged photos or check-ins. As a consequence, the news feed also becomes location-aware [7, 19], where the news items in the news feed are not only from the subscribed sources, but also relevant to the user's location. However, due to (a) the large number of the relevant news items generated by the users' sources/friends and (b) the limited screen viewing capability, a user may not be interested in all the relevant messages. For example, a desktop user may be interested in the top-50 news items, while a mobile user may be only interested in the top-10 news items. As a result, the existing news feed systems [23, 7, 19] opt to return only the top-k most recent ones. However, with such strict temporal order, a user may end up missing very relevant news items that are very close to the user location, yet, they are not very recent. Moreover, different users may have different tastes for the relevant news items: traveling users may interested in the nearby messages, while stationary users may interested in the recent messages. Thus, there is a strong motivation calling for a new way to produce more "personalized" location-aware news feeds for the user, which include the top-k most relevant news items considering a user's preference.

In this paper, we present GeoRank, an efficient location-aware news feed ranking system, which produces the news feed based on a ranking function considering: (a) the temporal proximity, where more recent news items are favored, (b) the spatial proximity between the location of the news items and the user, where closer news items are favored, and (c) a user defined preference parameter $0 \leq u.\omega \leq 1$, which reflects the relative importance of the temporal and spatial proximity. For example, a traveling user may set a smaller $u.\omega$ to get more geographically close news items, while a stationary user may set a higher $u.\omega$ to include more recent ones.

The goal of GeoRank can be abstracted as an aggregated top-k query from multiple input lists (i.e., the news sources) based on a ranking function. One way to realize the GeoRank is to retrieve the top-k most relevant news items from each of the subscribed news sources and perform a global top-k selection afterwards. However, this approach is extremely inefficient and may introduce significant response delay, because: (1) retrieving the top-k most relevant news items from each news source can be costly, especially when it possesses a large number of news items; and (2) a user may have subscribed to many sources (e.g., a typical user in Facebook follows 150 friends [81]), which results in evaluating a large number of top-k most relevant news items queries. As users may issue the news feed request repeatedly over the time, the user response time and the system efficiency are the main concerns. To this end, GeoRank avoids such costly approach by injecting the ranking function deep inside the news feed system. GeoRank employs spatial and temporal pruning techniques that not only avoid retrieving the messages that will not make it to the top-k news feed result, but also avoid evaluating the top-k queries to the news sources who will not contribute any messages to the news feed. In that way, GeoRank significantly reduces the user response time and system overhead.

Although injecting a top-k ranking function inside of query operations, especially those operations that read data from different lists, is a well studied problem (e.g., see [48] for a survey), unfortunately, existing techniques are not applicable to our problem due to the following four main reasons: (1) Most existing top-k techniques require that input lists are sorted on the attribute that contributes to the ranking function. Such requirement is not applicable to our case, as a news source will be subscribed by many users who want to get the news feed at different location. It is not possible to provide an universal ranked spatial order in advance. (2) Existing techniques focus on addressing one top-k query at a time. GeoRank is different, where each user in the social network poses a unique location-aware news feed query to multiple news sources upon logging on to GeoRank. So, different users may retrieve news items for the same input lists. Thus, GeoRank has the opportunity to exploit optimization and indexing techniques that will be shared by all users. (3) Existing techniques have the implicit assumption that number of items in the result (i.e., k) is significantly higher than the number of input lists, which could be only two or three lists. The environment, where GeoRank is applied, exhibits a completely opposite behavior, where a typical value of k could be 30 or 50, while the number of input lists has an average value of 150, e.g., the number of friends for an average Facebook user. Such property gives the room for different spatial and temporal pruning techniques. And (4) Most of the existing top-k techniques overlook the updates in the input list, where the input lists are static. However, in the GeoRannk, the news sources may update new messages continuously, where an efficient updating algorithm is also needed.

GeoRank consists of two main modules, namely, the *GeoRank query processor* and the *GeoRank message updater*. *GeoRank query processor* is mainly responsible for producing the top-k ranked location-aware news items. *GeoRank query processor* employs a two-stage pruning technique. On the first stage, the *query processor* module prunes those news sources (or friends) that will never contribute any news item to the news feed. On the second stage, the *query processor* module exploits spatial and temporal pruning techniques to avoid evaluating all the news items from each candidate news source, i.e., the sources that are not pruned from the first stage. The key for effective pruning techniques in the *query processor* module is the availability of pre-computed statistics that are well maintained for each user, while the user is offline.

On the other side, *GeoRank message updater* module does not really contribute to the answer of any top-k news feed requests. Instead, it is a background process with the sole responsibility of maintaining the set of pre-computed statistics used later by *query processor* module in all its pruning techniques. In a nutshell, GeoRank encounters a system overhead through its *message updater* module in continuously maintaining a set of statistics for each user. However, this offline system overhead is amortized by the savings in online query response time for each user's news feed request. *GeoRank message updater* module is mostly triggered by each submitted new message to check if it will affect any of the existing pre-computed statistics. The efficiency of the *message updater* module stems out from its ability to smartly point out those statistics that will be affected by every new message submitted to the system. Another trigger for the *GeoRank message updater* module is that the user follows a new source, where the *message updater* module will initialize the statistics for the relationship.

Extensive experimental results based on real data sets crawled from Twitter show the efficiency and scalability of GeoRank, along with the effectiveness of its pruning and optimization techniques used in both its *query processor* or *message updater* modules.

The rest of the chapter is organized as follows: Section 4.2 presents preliminaries in GeoRank. Section 4.3 gives an overview of the system. The details of GeoRank major modules, *query processor* and *message updater* are presented in Sections 4.4 and 4.5, respectively. Experimental results are given in Seciton4.6. Finally, Section 4.7 concludes the paper.

4.2 Preliminaries

4.2.1 Messages, Users, and News Feed

Location-based Messages. Similar to the existing social networking services, each message M from a user u has a timestamp $M.t$ indicating its issuing time. In addition, each message has an explicit geographical coordinate $M.loc$, indicating the location, which can be either the user's current location extracted from a GPS equipped device or the most related location inferred from the contents.

System Users. A user u in GeoRank maintains four main attributes: (1) the user location $u.loc$, which can either be the user current location or a fixed location set in the user profile. In GeoRank, a user's location is static, as in the most cases, users get the news feed frequently from certain locations, e.g., office or home. If a user has more than one location for getting the news feed, we consider her as two different users in the system; (2) a set of followers $u.\mathcal{F}$ that represents the set of other GeoRank users who have indicated their interests in any location-based messages posted by u; (3) a set of sources $u.\mathcal{S}$, that represents the set of other GeoRank users, where u has indicated his/her interest in any location-based messages posted by any of them; and (4) a preference parameter $0 \leq u.\omega \leq 1$, that weights the user preference towards spatial and temporal proximity, described in Section 4.2.2.

Rank Aware Location-based News Feed. Once the user u logs on to GeoRank, u receives news feed messages that are: (a) issued by one of the users in u's source list, $u.\mathcal{S}$, and (b) among the top-k highest scores, according to the message ranking function, described in Section 4.2.2, that considers user location $u.loc$, message location $M.loc$, message time $M.t$, user log on time $u.t$ and user preference parameter $u.\omega$. k is a system wide parameter indicating the total number of messages that will be delivered. For example, a typical value of k is 50 for desktop users and 30 for a mobile devices. During a logging session, a user u may post location-based messages that may be seen later by her follower in $u.\mathcal{F}$.

4.2.2 Message Ranking Function

GeoRank employs a ranking function $Ranking(u, M)$ that gives a relevant score of a message M to a user u. The higher the score the more relevance is the message. The ranking function encapsulates both the temporal and spatial proximity of M to u. Since spatial and temporal proximity have two different domains, i.e., distance in miles and time in hours, the ranking function normalizes the spatial and temporal proximity to have a normalized domain score from 0 to a maximum score *Max*. The ranking function also employs a personalized preference parameter $0 \leq u.\omega \leq 1$ that weights the relative importance to a user. Formally, the message ranking function can be represented as:

$$\begin{aligned} Ranking(u, M) = u.\omega \times \textit{NormTemporal}(u.t, M.t) \\ +(1 - u.\omega) \times \textit{NormSpatial}(u.loc, M.loc) \end{aligned} \tag{4.1}$$

A higher value for the user preference parameter $u.\omega$ indicates a higher weight to the temporal proximity than the spatial proximity. At one extreme, setting $u.\omega$ to 0 indicates that the user only cares about the closest k messages, among the ones issued by $u_j \in u.\mathcal{S}$, which is similar to the results of k nearest neighbors (*KNN* queries) [58]. On the other extreme, setting $u.\omega$ to 1 indicates that the user only cares about the most recent messages, which is equivalent to the traditional news feed function [23].

In the mean time, *NormTemporal*$(u.t, M.t)$ is a normalization function that normalizes the time differences between the user logging on time $u.t$, which is NOW, and the message issue time $M.t$ to scale from 0 to the maximum scoring value *Max*. The temporal normalization function decreases linearly within a range of a predetermined temporal boundary T, where the score of *Max* is given when there is no time difference between the log on time and the message issuing time, while a score of 0 is given to those messages that are older by T time units or more from the log-on time. Similarly, *NormSpatial*$(u.loc, M.loc)$ is a normalization function that normalizes the Euclidean space difference between the user location $u.loc$ and the message location $M.loc$ to scale from 0 to a maximum value *Max*. The spatial normalization function also decreases linearly, up to a predetermined spatial boundary S.

4.2.3 Problem Definition

Our problem in GeoRank can be formulated as: "*Given a set of users $\mathcal{U}$ in the system, where each user u has a list of followers $u.\mathcal{F}$, a list of sources $u.\mathcal{S}$, and a preference parameter $u.\omega$. Once a user u logs on the system, GeoRank finds the top-k most relevant news items, based on her current location $u.loc$, log on time $u.t$, and the preference parameter $u.\omega$ from her sources*".

The main contribution of GeoRank is not to create an alternative ranking method for news feed, but to improve its efficiency. In that way, users can enjoy news feed with short response times, while the system is not overwhelmed.

4.3 GeoRank System Overview

This section presents an overview of GeoRank system, which includes the system architecture, along with its underlying data structures that will be used throughout the rest of this paper.

4.3.1 System Architecture

Figure 4.1 depicts the system architecture of GeoRank, which is composed of two main modules, namely, the *GeoRank query processor* and the *GeoRank message updater*, briefly described below:

GeoRank query processor. The *query processor* module is automatically triggered for a user u, once u logs on to GeoRank system. The *query processor* module first consults the source list $u.\mathcal{S}$ to find and rank the relevant messages, using the user preference parameter $u.\omega$ and the ranking function $Ranking(u, M)$ (Arrows 1 and 2 in Figure 4.1). The output of the *query processor* module is a set of k messages as the top-k highest ranked messages among all the messages posted from the user's subscribed sources $u.\mathcal{S}$ (Arrows 3 and 4 in Figure 4.1). The *query processor* module employs a set of smart pruning techniques to avoid scanning all possible sources and messages. In addition, it relies on pre-computed statistics that significantly

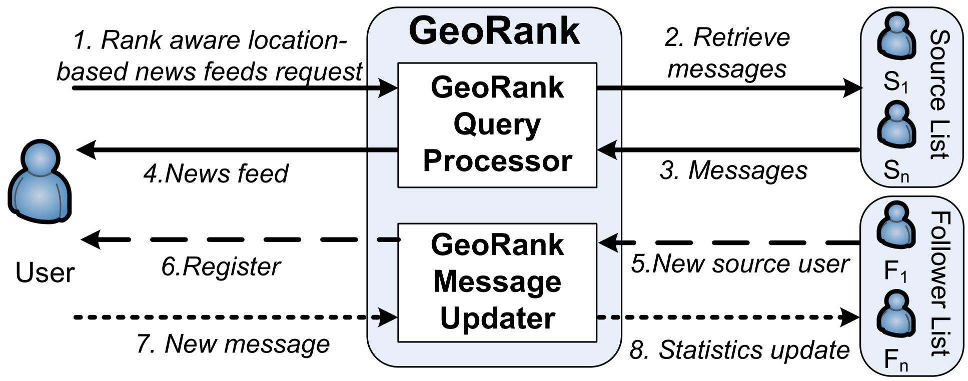


Figure 4.1: GeoRank System Architecture.

enhance the query performance. Details of the *query processor* module will be described later in Section 4.4.

GeoRank message updater. The main purpose of the *message updater* module is to track the set of statistics used later by the *query processor* module for improving the efficiency. The statistics include the most relevant (or highest ranked) messages for u from each user in its sources $u.\mathcal{S}$. The *message updater* module is triggered by: (1) Adding a new source user, where user u_f follows the message updates from user u (Arrow 5 in Figure 4.1). In this case, we initialize the most relevant message, among the ones posted from u, for u_f, along with with few other statistics; and (2) A new message M posted by user u (Arrow 7 in Figure 4.1). In this case, we update the statistics for the user's followers in $u.\mathcal{F}$, with internally used statistics. It is important to note that the *message updater* module does not directly contribute to the answer. Instead, it is a process running in the background to facilitate the mission of the *query processor* module, whenever called. Details of the *message updater* module will be described in Section 4.5.

4.3.2 Data Structure

In addition to the typical user data for each user u that includes the user id $u.id$ and location $u.loc$, GeoRank maintains the following data structure for each user u:

A preference parameter ($u.\omega$), which gives the user preference towards the spatial or temporal domains, as described in the ranking function in Section 4.2.2.

The source list ($u.\mathcal{S}$), as the set of other users that u is interested to receive messages from.
The follower list ($u.\mathcal{F}$), as the set of other users who are interested in receiving messages from u.
List of posted location-based messages ($u.\mathcal{M}$), as the list of prior posted messages from user u. Each message $M \in u.\mathcal{M}$ has a location $M.loc$ and timestamp $M.t$.
Spatial grid index ($u.\mathcal{G}$)**,** as the underlying spatial index structure, which consists of $n \times n$ equal area grid cells. A grid cell $C \in u.\mathcal{G}$ includes all the user followers $u.\mathcal{F}$, sources $u.\mathcal{S}$, and posted massages $u.\mathcal{M}$, whose locations fall within the cell area boundary.

4.4 GeoRank Query Processor

The query processor is responsible for providing the top-k relevant news feed for user u. A straightforward way to support this functionality is to decompose a user's news feed request into $|u.\mathcal{S}|$ top-k queries, as one query for each source user $u_s \in u.\mathcal{S}$. Then, the final result is assembled by aggregating the overall top-k ranked messages among the ones retrieved from all the source users. The *query processor* module in GeoRank avoids such naive (and prohibitively expensive) solution through a two-step approach that is based on two main concepts: (a) It is not necessary to check all the source users $u_s \in u.\mathcal{S}$ for the relevant messages, as it is most likely that $k < |u.\mathcal{S}|$. For example, in Facebook, an average user has 150 friends (i.e., sources), yet k may be set to 50. This means that we can get all the top-50 ranked messages from at most 50 friends, and (b) It is not necessary to return k messages from each checked source user $u_s \in u.\mathcal{S}$, as most of such returned messages will not qualify for the final top-k answers.

In order to exploit these two concepts, GeoRank query processor follows a two-step approach. The first step, termed *candidate sources selection* (Section 4.4.2), exploits the first concept to early prune a large number of source users, who will never contribute to any of the top-k news items. Non-pruned sources are considered as candidate sources and checked further in the second step. The second step, termed *news feed aggregation* (Section 4.4.3), exploits the second concept by using spatial and temporal pruning techniques, along with an early termination condition, to minimize the number of retrieved messages to produce the top-k ranked news feed.

Algorithm 2 gives the pseudo code for *GeoRank query processor*, with its two main steps. The input to the algorithm is the user location $u.loc$, preference parameter $u.\omega$, source list $u.\mathcal{S}$, list of the most relevant messages $u.\mathcal{R}$, grid index structure $u.\mathcal{G}$, and log on time $u.t$, which is set as NOW. Notice that we do not need anything related to the set of user's followers $u.\mathcal{F}$ within the *query processing* module, as none of them will contribute to the messages that the user u will receive in her news feed. The output of the algorithm is the set of highest ranked k location-based messages based on their spatial and temporal relevance to the user u according to u's preference parameter $u.\omega$ and the message ranking function.

4.4.1 Additional Data Structure

In addition to the data structure described in Section 4.3.2, the *query processing* module maintains the following data structure:

List of most relevant messages from the sources ($u.\mathcal{R}$), as one message from each source user $u_s \in u.\mathcal{S}$. A most relevant message R_s, posted by a source user u_s, is pre-computed and continuously maintained in GeoRank. R_s represents the highest ranked message posted from the source user u_s with respect to user u according to the user's message ranking function and preference parameter $u.\omega$. This is a key structure in the GeoRank query processor as it plays a major role in all the pruning techniques. It is important to note that the *query processor* module in GeoRank just uses this key data structure to produce the results. However, the computation and the continuous maintenance of this data structure is all done by the *GeoRank message updater* module, described in Section 4.5.

4.4.2 STEP 1: Candidate Sources Selection

The *candidate sources selection* step aims to exploit the fact that the number of messages k is highly likely to be less than the number of source users, i.e., $k < |u.\mathcal{S}|$. This means that at least $|u.\mathcal{S}| - k$ sources will never contribute to the top-k news feed. The objective of this step is to find out these $|u.\mathcal{S}| - k$ sources and exclude them, along with their messages, from any further consideration.

Main idea. The main idea of this step relies on the list of the most relevant messages $u.\mathcal{R}$ to prune $|u.\mathcal{S}| - k$ sources. Since this list includes exactly one message from each source user, then we can just take the highest k ranked messages, and only consider these sources users. It means that any source user that does not contribute to the highest k messages in $u.\mathcal{R}$ will never contribute any message to the news feed. The reason is that we can easily create a news feed with k messages (aggregating the top-k ranked messages from the list of the most relevant messages $u.\mathcal{R}$), where all the messages there have higher ranking scores than the highest ranked ones from the excluded source users. As a result, if we can get these top-k ranked source users based on their most relevant message scores, we can easily avoid a significant amount of computations in the news feed processing. However, producing such top-k list based on the most relevant messages from the source users is not trivial, as the only thing that we can pre-compute is the source's most relevant message itself. The reason we cannot store the message score is that the ranking score is mainly depend on the user's log-on time (i.e., $u.t$), which is needed in the temporal normalization, i.e., *NormTemporal*$(u.t, M.t)$, as a part of the message ranking function (described in Section 4.2), and not known as a priori. This calls for a online computing for the *current score* of each message in $u.\mathcal{R}$, once u is logged on, as only then, we know about $u.t$ and the score for temporal normalization. The correctness of this approach comes from the fact that even without knowing the score of each message $R_s \in u.\mathcal{R}$, we are still confident that this R_s has the highest ranking score among all messages produced by the source user u_s. We will elaborate more on it, when discussing the *message updater* module (Section 4.5), which ensures the sanity of $u.\mathcal{R}$.

Algorithm. Lines 2 to 13 in Algorithm 2 gives the pseudo code of the *candidate sources selection* step. The algorithm iterates over all the messages in $u.\mathcal{R}$, while calculating the score of each message based on the user's location, log-on time, and the underlying message ranking function. The source users that may contribute to the top-k highest ranked messages are stored, along with their ranking scores, in the list $CandList$, for the further processing. Meanwhile, *minscore* is set as the minimum score we have in $CandList$.

4.4.3 STEP 2: News Feed Aggregation

Given a set of *candidate sources* (i.e., $CandList$), produced from Step 1, a naive way to produce the top-k news feed for a user is to just get the local top-k messages from each source $u_s \in CandList$, and then proceed to find the global top-k messages across all the candidate sources. The *news feed aggregation* step aims to avoid such naive way by: (a) minimizing the number of retrieved items from each source $u_s \in CandList$ using spatial and temporal pruning techniques, and (b) avoiding checking all the sources $u_s \in CandList$ through an early termination condition. The output of this step is the requested top-k messages for user u.

Main idea. The main idea of the *news feed aggregation* step is to use the current minimum

Algorithm 2 GeoRank Query Processing

Input: user location $u.loc$, preference parameter $u.\omega$, source list $u.\mathcal{S}$, most relevant messages $u.\mathcal{R}$, grid index $u.\mathcal{G}$, and log on time $u.t$.
Output: Top-k highest ranked messages.

//Step 1. Candidate source selection
minscore $\leftarrow$ 0; *count* $\leftarrow$ 0; $CandList \leftarrow \phi$
for each message $M_i \in u.\mathcal{R}$ **do**
  score $\leftarrow u.\omega \times$ *NormTemporal*$(u.t, M_i.t)$ + (1-$u.\omega$) $\times$ *NormSpatial*$(u.loc, M_i.loc)$
  if *score* $>$ *minscore* **OR** $count < k$ **then**
    $CandList \leftarrow CandList \cup (u_i$, *score*)
    if $count > k$ **then**
      Remove (u_{min}, *minscore*)
    end if
    minscore $\leftarrow$ minimum score in $CandList$
    count $\leftarrow$ *count* + 1
  end if
end for
//Step 2. GeoRank news feed aggregation
$N \leftarrow k$; *Result* $\leftarrow$ All messages in $CandList$
for each source $u_i \in CandList$, ordered by ranking score **do**
  $\mathcal{D} \leftarrow$ *NormSpatial*$^{-1}(\frac{minscore - u.\omega \times Max}{1-u.\omega})$
  $\mathcal{T} \leftarrow$ *NormTemporal*$^{-1}(\frac{minscore-(1-u.\omega)\times Max}{u.\omega})$
  Result $\leftarrow$ *Result* $\cup$ top-N messages from u_i within $\mathcal{D}$ & $\mathcal{T}$ (retrieved from $u.\mathcal{G}$, ranked by score)
  $N \leftarrow k-$ number of items in *Result* with a score higher than u_{i+1}.*score*
  if $N \leq 0$ **then**
    Return top-k messages in *Result* as the final result
  end if
  minscore $\leftarrow$ The score of the kth item in *Result*
end for
Return top-k messages in *Result* as the final answer

score (*minscore*) of all available message ranking scores in $CandList$, to compute both spatial and temporal boundaries that limit the number of messages retrieved from each candidate source. In addition, we incrementally maintain a set of valid candidate messages along with their *minscore*, which is used to update the spatio-temporal boundaries and limit the number of further processed messages from each candidate source user. This can be summarized in the following three ideas:

- *Spatial boundary*. Given that the kth highest ranked message we have so far for u has the score *minscore*, then for a message M to make it among the top-k messages for u, M has to have a higher score than *minscore*, i.e., per Equation 4.1, $u.\omega$ $\times$ *NormTemporal*$(u.t,$

$M.t$) + (1- $u.\omega$) $\times$ *NormSpatial*($u.loc$, $M.loc$) $>$ *minscore*. In order to get a spatial boundary of where the message location $M.loc$ should be, we assume that M has the highest possible temporal score *Max*. In this case, in order for M to make it to the highest top-k messages, M has to be located inside the spatial area $\mathcal{D}$, as follows:

$$\mathcal{D} = \mathit{NormSpatial}^{-1}(\frac{\mathit{minscore} - u.\omega \times \mathit{Max}}{1 - u.\omega}), \tag{4.2}$$

where *NormSpatial*$^{-1}()$ is the inverse function of the spatial normalization in the user's message ranking function. This means that if a message M is located outside of the area $\mathcal{D}$, M will have no chance in having a higher score than *minscore*, hence will not make it to the top-k items.

- *Temporal boundary*. Similar to the case of determining a spatial boundary, in order to get a temporal boundary of when the message was posted, we assume that M has the highest possible spatial score *Max*. In this case, in order for M to make it to the highest top-k messages, M has to be posted in the last $\mathcal{T}$ time units, computed as follows:

$$\mathcal{T} = \mathit{NormTemporal}^{-1}(\frac{\mathit{minscore} - (1 - u.\omega) \times \mathit{Max}}{u.\omega}), \tag{4.3}$$

where *NormTemporal*$^{-1}()$ is the inverse function of the temporal normalization in the user's message ranking function. This means that if M was posted older than $\mathcal{T}$ time units, M will have no chance in making it to the top-k items.

- *Number boundary*. We start by the objective of getting the top-k messages. Then, as we visit each source user in the candidate sources, $CandList$, we start to confirm that a certain number of message, x, will definitely be among the top-k ones. In this case, for the next source user to visit, we only look for retrieving at most $|k - x|$ messages. As we keep lowering our number boundary, we early terminate our search when the number boundary reaches 0.

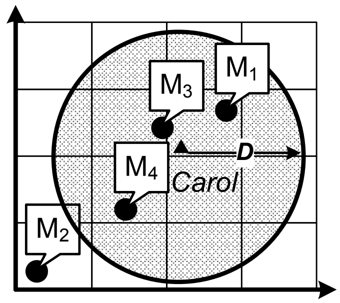
Algorithm. Lines 15 to 26 in Algorithm 2 gives the pseudo code of the *news feed aggregation* step. We initially set our number boundary N as k, and the output result set as the list of messages in $CandList$. Then, we iterate over each source $u_i \in CandList$, in descending order of their most relevant message scores, as the source user with higher score has a higher chance to contribute messages to the news feed result. For each source, we calculate both the spatial and temporal boundaries $\mathcal{D}$ and $\mathcal{T}$, per Equations 4.2 and 4.3, respectively. Then, we exploit the spatial grid index structure $u.\mathcal{G}$ to retrieve the highest N ranked messages that lie within our spatial and temporal boundaries from source u_i. The retrieved messages, which could be at most N messages, are inserted to our current result set. Our number boundary N is set by subtracting k with the number of messages in the current result set that has a higher score than the score of the next source user. This is mainly as these messages are guaranteed to be in the top-k result. If the number boundary becomes lower than or equal to zero, we just terminate the algorithm and return the top-k messages in the current result set as the news feed. Otherwise, we set the value of the new minimum score (*minscore*) as the score of the kth item in the current result set, and proceed to check on the next source u_{i+1}. The algorithm continues till we either have our number boundary $N \leq 0$, or go through all the sources in the $CandList$. The latter case corresponds to the unlikely case, where each source $u_i \in CandList$ contributes exactly one message to the user's news feed.

Example. *Carol* requests her top-3 location-based news feed at time 20:00. Then, assuming that the *candidate sources selection* step returns three candidate sources, i.e., *Alice*, *David*, and *Bob*, with the their most relevant message scores of 8.5, 7.0, and 6.7, respectively. Thus, the minimum score (*minscore*) is set to 6.7. Figure 4.2 illustrates the *news feed aggregation* step that will be executed for *Carol*, where we go through the three candidate sources based on their scoring order, i.e., we first start with user *Alice*. Figure 4.2a depicts *Alice*'s four messages, with their issuing times and locations, marked as black dots on her grid spatial index. Assuming that we have calculated the temporal boundary $\mathcal{T}$ using Equation 4.3 to be 15:00 and the spatial boundary $\mathcal{D}$ using Equation 4.2 to be the circle around *Carol*'s location in *Alice* spatial index. Since *Alice* is the first user to check for, our number boundary N is initialized by k=3. Based on the temporal, spatial, and number boundaries, we only need to retrieve two messages from *Alice*, namely, M_4 and M_3, with scores 8.5 and 6.9, respectively. Among M_4 and M_3, we know for sure that M_4 will make it to the final answer as it scores higher than the most relevant message score of the next source user, i.e., *David*. However, we are not yet sure about the fate of M_3. Since M_3 has a lower score than the highest scored message from *David*, then there is a probability that *David* may have two messages higher than M_3. With this, we update our number boundary to be $N = 2$, indicating that we are still looking for two more messages from *David*. Also, the minimum score is updated to 6.9 as the kth ranked message we have so far, which is M_3.

Figure 4.2b depicts *David*'s messages. With the updated minimum score, the temporal

MSG	Time	Score
Alice.M_4	19:30	8.5
Alice.M_3	15:00	6.9
Alice.M_2	9:00	-
Alice.M_1	8:00	-

Alice's Messages



Alice's Spatial Index

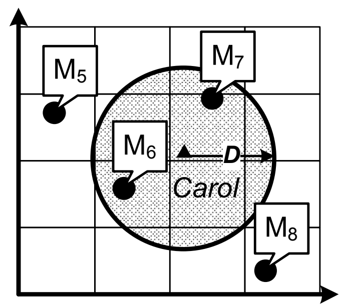
MSG	Score
Alice.M_4	8.5
David.M_7	7.0
Alice.M_3	6.9
minscore	6.9

Carol's Result Set

(a) GeoRank News Feed Aggregation in source user *Alice*

MSG	Time	Score
David.M_8	19:30	-
David.M_7	17:00	7.0
David.M_6	16:40	6.2
David.M_5	15:00	-

David's Messages



David's Spatial Index

MSG	Score
Alice.M_4	8.5
David.M_7	7.0
Alice.M_3	6.9
minscore	6.9

Carol's Result Set

(b) GeoRank News Feed Aggregation in source user *David*

Figure 4.2: Example of GeoRank News Feed Aggregation Step.

boundary $\mathcal{T}$ becomes tighter as 16:00 while the spatial boundary $\mathcal{D}$ is depicted by a smaller circle. Only two messages satisfy the new spatial, temporal, and number boundaries, namely, M_7 and M_6 with scores 7.0 and 6.2, respectively. With this, we know that for sure M_7 and M_3 will be in the final result, as both of them score higher than 6.7, which is the highest message score from *Bob*. So, we just update our number boundary to be 0. As this is our stopping criteria, we terminate the algorithm without visiting *Bob*, where news feeds include M_4, M_7, and M_3, in order.

4.5 GeoRank Message Updater

As discussed in the previous section, the *query processor* module mainly relies on the list of most relevant messages, $u.\mathcal{R}$, in all its pruning techniques. However, the *query processor* module has dealt with this list as a given input, and has nothing to do with computing and maintaining it. In this section, we discuss *GeoRank message updater* module, where its main purpose is to ensure the sanity and accuracy of the list $u.\mathcal{R}$. For a user u, computing the most relevant message R_s from a source $u_s \in u.\mathcal{S}$ includes two steps, *initialization* and *maintenance*. The *initialization* step takes place when a new source user u_s is followed by the user u. Then, u will need to compute an initial value of R_s, based on the messages from the source user u_s, i.e., $u_s.\mathcal{M}$. The *maintenance* step is triggered with each new message posted from the source user $u_s \in u.\mathcal{S}$, where we will need to check if the new message has a higher score to u than the current most relevant message from u_s.

The challenge here comes from the fact that each new message from a source user $u_s \in u.\mathcal{S}$ is not only relevant to u, but, it is also relevant to all those users that consider u_s as one of their sources, i.e., all the users in the follower list $u_s.\mathcal{F}$. A straightforward solution can go as follows: Once a source user u_s submits a new message, we scan all the followers $u_f \in u_s.\mathcal{F}$ to check if we need to update the most relevant message with the new message for any of these users. This straightforward solution can be extremely inefficient, where users may have large number of followers and produce large number of messages, e.g., according to the Facebook statistics [81], an average user has 150 friends and creates over 90 pieces of content each month. Doing an exhaustive search of every posted message over every user's follower may be prohibitively expensive. *GeoRank message updater* module avoids such prohibitively expensive operations by employing spatial filters that limit the number of followers to check for. This is done by associating a set of $|u.\mathcal{F}|$ monitoring areas for user u, as one monitoring area per follower $u_f \in u.\mathcal{F}$. Then, whenever u posts a new message M at location $M.loc$, we do not need to check for all u's followers. Instead, we only check on those followers, whose monitoring areas overlaps with the new message location $M.loc$.

4.5.1 Additional Data Structures

In addition to the data structures described in Section 4.3.2, the *message updater* module maintains the following data structure:

List of monitoring areas for the followers ($u.\mathcal{A}$), as one monitoring area for each user's follower $u_f \in u.\mathcal{F}$. A monitoring area A_f is initialized and maintained by the *message updater* module to significantly reduce the computational costs for updating the list of most relevant messages for that follower $u_f.\mathcal{R}$. A monitoring area A_f basically says that in order for a new message M from user u to make it to the list of most relevant messages for the follower u_f, then, M has to be located inside A_f. All the follower's monitoring areas are laid out in the user's spatial grid index structure, $u.\mathcal{G}$.

4.5.2 Initialization: New Source Update

Whenever a user u_1 decides to follow the message updates from another user u_2, u_2 is added to the list of source users of u_1, i.e., $u_1.\mathcal{S} = u_1.\mathcal{S} \cup u_2$, and, at the meanwhile, u_1 is added to the list of followers of u_2, i.e., $u_2.\mathcal{F} = u_2.\mathcal{F} \cup u_1$. Then, the *message updater* module needs to take two actions: (1) initialize the most relevant message $R_2 \in u_1.\mathcal{R}$, where R_2 is the most relevant (highest ranked) one to user u_1, among all other messages posted from u_2; and (2) initialize the monitoring area $A_1 \in u_2.\mathcal{A}$ in user u_2, which says that any newly posted message from u_2 that is located outside area A_1 will never make it to the list of most relevant messages of u_1, $u_1.\mathcal{R}$. The two initialization actions are outlined below:

Action 1: Initializing the most relevant message $R_2 \in u_1.\mathcal{R}$. To perform this action, we need to find out the most relevant message to u_1, among all the ones posted by $u2$. Since all posted messages by u_2, $u_2.\mathcal{M}$, are sorted by their issuing times, we just scan the list $u_2.\mathcal{M}$, and calculate the score of each message based on the temporal order. We first consider the latest message as the most relevant one. Assuming that the score of this message is *MaxRelScore*, then we update the most relevant message whenever we find a message with a higher score than *MaxRelScore*, and accordingly adjust the value of *MaxRelScore*. Notice that a message M_2 with a lower temporal score than message M_1 can still have a higher overall score than M_1, if M_2 has a higher spatial score than M_1. We early terminate our scanning process if the next message to visit in $u_2.\mathcal{M}$ cannot score higher than *MaxRelScore*, regardless of its proximity to the location of u_1. In this case, we know that none of the subsequent messages will score higher than *MaxRelScore*, also regardless of its proximity to u_1. To judge on this early termination procedure, we will assume that the next message to visit M_n has the maximum spatial score *Max*. Then, M_n score will be: *Ranking*(u_1,M_n) = $u_1.\omega \times$ *NormTemporal*(u_1,M_n) + $(1 - u_1.\omega)$ $\times$ *Max*. This means that we can early terminate, if this score is less than *MaxRelScore*, i.e., the temporal score of the next message M_n is less than $\frac{MaxRelScore-(1-u_1.\omega)\times Max}{u_1.\omega}$. Finally, it is important to note that we only store the most relevant message R_2 in $u_1.\mathcal{R}$ without its score *MaxRelScore*. This is mainly because this score will be irrelevant when time advances, as its value is based on the temporal score, which can only be computed with the current time.

Action 2: Initializing the monitoring range area $A_1 \in u_2.\mathcal{A}$. To perform this action, we will need to find an area A_1 such that if a new incoming message M from u_2 is located outside A_1, then we can safely conclude that M will never make it to the list of most relevant messages for user u_1, $u_1.\mathcal{R}$. We will use the most relevant message R_2, computed in Action 1, as an estimation of the initial value of A_1. We can safely say that if a new incoming message M has less score than that of R_2, then, M will have no chance in replacing R_2. Although we know the current exact score of R_2, we cannot rely on this score when comparing it with the score of the new message M, as the score of R_2 decays over time, where its issuing time becomes further. To make R_2 and M comparable, we use a very conservative estimation of the score of R_2 at the time when u_2's new message M is issued. The conservative estimation is obtained by considering that the temporal score of R_2 has decayed to its lowest possible value 0. In this case, the minimum possible score of R_2 is *MinScore* = (1-$u_1.\omega$) $\times$ *NormSpatial*($u_1.loc$,$R_2.loc$). Then, with another conservative assumption, we assume the incoming message M has the highest possible temporal score *Max*. Then, for M to have a higher score than R_2, the following criteria should hold: *Ranking*(u_1,M) $>$ *MinScore*, i.e., $u_1.\omega \times$ *Max* + $(1-u_1.\omega) \times$ NormSpatial(u_1,M) $>$ *MinScore*. This means that A_1 is a circular centered at $u_1.loc$ with a radius $A_1.r$, computed per the following equation:

$$A_1.r = \mathit{NormSpatial}^{-1}\left(\frac{\mathit{MinScore} - u_1.\omega \times \mathit{Max}}{1 - u_1.\omega}\right), \tag{4.4}$$

Finally, the computed monitoring area A_1 is laid out on the source user's grid spatial index $u_2.\mathcal{G}$.

4.5.3 Maintenance: New Message Update

Whenever a user u posts a new message M, u needs to check if M will affect any of the most relevant messages R_u for the followers, $u.\mathcal{F}$. If so, we update the most relevant messages accordingly, along with their corresponding monitoring areas stored at u.

Main idea. The main idea of the maintenance step is to exploit the grid spatial index structure

Algorithm 3 GeoRank Message Updating

Input: A message M posted by user u

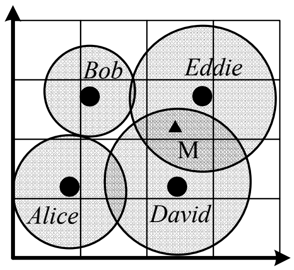
```
1: AffectedList ← φ
2: C ← The grid cell in u.𝒢 that includes M.loc
3: for each monitoring area A_f ∈ u.𝒜 located in cell C do
4:    if M.loc is inside A_f then
5:       AffectedList ← AffectedList ∪ u_f
6:    end if
7: end for
8: for each follower u_f ∈ AffectedList do
9:    R_u ← Retrieve u's most relevant message to u_f from u_f.ℛ
10:   MostRelScore ← u_f.ω × NormTemporal(NOW,R_u.time) + (1-u_f.ω) × NormSpatial(u_f.loc,R_u.loc)
11:   NewMsgScore ← u_f.ω × Max + (1-u_f.ω) × NormSpatial(u_f.loc,M.loc)
12:   if NewMsgScore > MostRelScore then
13:      R_u ← M in u_f.ℛ
14:      MinScore ← (1-u_f.ω) × NormSpatial(u_f.loc,M.loc)
15:      A_f.r ← NormSpatial^{-1}((MinScore − u_f.ω × (Max)) / (1 − u_f.ω))
16:   end if
17: end for
```

maintained at u, $u.\mathcal{G}$, to early prune followers that will not be affected by the new message M. Such early pruning avoids an exhaustive scan over all followers of u. For those followers who are not pruned, we still do an extra check to see if M actually affects their most relevant message from u, as we used the conservative way to calculate the monitoring areas. If this is the case, we update the most relevant message for the follower with the new message. Finally, we use the updated most relevant message to calculate a new monitoring area for such followers, in a similar way to the initialization module discussed in section 4.5.2.

Algorithm. Algorithm 3 gives the pseudo code for the *message updater* module upon receiving a new message M from user u. The pseudo code has two main steps:

(1) Finding out the list of followers that may be affected by the new message M (Lines 1 to 7 in Algorithm 3). We do so by first locating the cell C in $u.\mathcal{G}$ that includes the message location $M.loc$. Then, for any follower u_f whose monitoring area A_f is registered in cell C, we do an extra check to see if $M.loc$ is located inside A_f. If this is the case, u_f will be added in the list of affected followers.

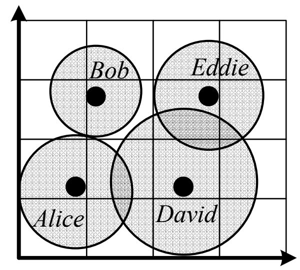
(2) For each affected follower u_f, we update its most relevant message R_u with the new



(a) *Affect Follower Selection*

Follower	R_u	Most Rel	New Msg
Alice	M_3	-	-
Bob	M_4	-	-
David	M_2	7.5	7.3
Eddie	M	7.1	7.9

(b) *Most Relevant Message Update*



(c) *Monitoring Range Update*

Figure 4.3: Example of Message Update in GeoRank.

message M posted from u, if needed. In case that the message update takes place, we also update the monitoring area A_f at user u (Lines 8 to 17 in Algorithm 3). We do so by doing the following for each user u_f in the list of affected followers: (a) We retrieve R_u from the list $u_f.\mathcal{R}$ as the current most relevant message from user u to follower u_f, (b) As we do not have the score of R_u, we will need to calculate R_u score (i.e., *MostRelScore*) based on the current time and the follower location. It is important to note that we could not store this score with R_u as it decays over time, and has to be recomputed with every time instance. Thus, we opt to compute it only when needed, (c) We compute the score of the new message M (i.e., *NewMsgScore*) considering that the temporal score is of a maximum value as M is just posted now, while the spatial score is computed based on the proximity of message location to the follower's location $u_f.loc$, and (d) We compare the score of the most relevant message (*MostRelScore*) against that of the new message M (*NewMsgScore*). If the new message has a lower score, we just do nothing, as the new message will not affect anything in our maintained data structure. On the other hand, if the new message has a higher score than that of the currently most relevant message, we first replace the currently most relevant message R_u by the new message M. Then, in a similar way to what we have done in section 4.5.2, we calculate the minimum possible score of M using only its spatial score, and use this value to update the radius of the monitoring area A_f for that follower in the user's spatial index $u.\mathcal{G}$.

Example. Figure 4.3 gives an example of the maintenance algorithm in *GeoRank message updater*. Figure 4.3a depicts the spatial grid index at user *Carol*, where she maintains four circular monitoring areas that correspond to her four followers *Alice*, *Bob*, *David*, and *Eddie*, along with the message M posted from *Carol* (depicted by a small triangle). As the location of message M is located outside the monitoring areas of *Alice* and *Bob*, we just early prune these two followers, as we know for sure that their most relevant message will not be affected by M. Figure 4.3b depicts the further actions taken on the remaining users, *David* and *Eddie*, where we calculate their scores of the new message M based on their preference parameters and locations, which, ended up to be 7.3 and 7.9, respectively. Assume that the prior most relevant scores at *David* and *Eddie* are 7.5 and 7.1, respectively. By comparing the new computed scores for M by the old most relevant scores, we find that M has a lower score than what *David* already has, so, we just exclude *David* from any further considerations. In the mean time, we find that M actually gives a higher score that what *Eddie* already has. In this case, we do two actions: (1) Update the most relevant message R_{carol} at *Eddie* to be M, and (2) Update the monitoring area of *Eddie* to be tighter based on the new message M (Figure 4.3c).

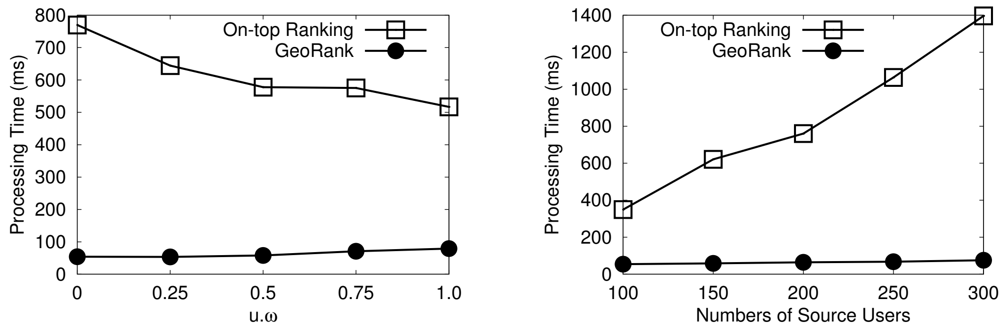
4.6 Experimental Evaluation

Experimental evaluations of GeoRank are based on an actual system implementation in PostgreSQL database management system [77]. Experiments are based on a set of 10 Million geo-tagged Twitter messages (i.e., tweets) issued within the state of Minnesota, US, generated as follows: First, we crawled the twitter message data via Twitter Search API[1] for one week. Then, we got $\approx$ 650K distinct geo-tagged tweets, where the geographical information is represented as either a semantic location, e.g., a city name or latitude/longitude coordinates. In the former case, we use Google GeoCoding API[2] to convert into latitude/longitude coordinates. We make use of these real 650K tweets to get to know the real spatial and temporal distributions the tweets. Finally, we generate synthetic 10,000 GeoRank users, where each user is associated with 1,000 geo-tagged tweets. The locations and issuing times of all the 10 Million messages mimic the spatial and temporal distributions of our crawled real tweets. Also, locations of the 10,000 users are static and set randomly based on the locations of the real tweets.

Mostly taken from Facebook statistics [81], and unless mentioned otherwise, each user is following an average of 150 users (i.e., sources) and is also followed by another 150 users (i.e., followers), randomly picked from the user set. We set the default k as 30, which means the

[1] Twitter Search API: http://search.twitter.com.

[2] Google GeoCode:http://maps.googleapis.com/maps/api/geocode/



(a) Different User Preferences. (b) Different Source Users.

Figure 4.4: Compare With "On-Top" Approach.

user likes to see the top-30 messages in the news feeds. We use a simple ranking function, where *Max* is set to 10, and every one mile or one hour is equivalent to 0.1. The default user preference parameter ω is set to 0.5. Finally, a 10×10 grid structure is constructed for each user to index message locations and the monitoring areas. All experiments were evaluated on a server computer with Intel Core 2 Quad CPU 2.83GHz processor and 8 GB RAM running Ubuntu Linux 10.04.

4.6.1 GeoRank Overall Performance

In this section, we compare the overall performance of GeoRank against the simple *on-top* approach. In the *on-top* approach, we issue one top-k query to each source user and aggregate the top-k results to produce the news feed. We still optimize in the *on-top* using the temporal order of the messages for the early termination.

Figure 4.4a gives the performance of both approaches when varying the user preference parameter $u.\omega$ from 0 to 1. GeoRank consistently gives 6 to 10 times better performance than the *on-top* approach. This is mainly because GeoRank reduces the number of evaluated source users, i.e., we do not have to check on each source user, and also reduce the number of processed messages at each source. Another thing to note is that with the increasing of $u.\omega$, the *on-top* approach gets a better performance. This is because the temporal early termination technique is more effective with the more weight to the temporal domain. This is not the case in GeoRank as it gives the same good performance for all ω.

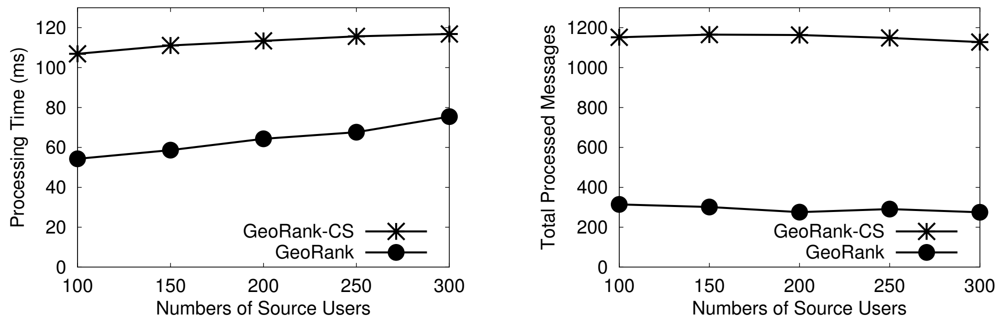
Figure 4.4b gives the performance, when varying the number of source users from 100 to 300 to simulate the users following different numbers of sources. In the experiment, each user follows randomly selected source users and we evaluate the average news feed processing time. As shown in the figure, GeoRank scales up well with the increase of the number of source users, while the performance of the *on-top* solution deteriorates significantly. Moreover, the performance gains by GeoRank increases significantly, when the querying user follows more source users. Especially for the case of 300 sources, GeoRank gives 28 times better performance than the *on-top* approach. The main reason is because of the pruning power at GeoRank avoiding all the disqualified source users, while *on-top* approach still needs to check on all source users. Essentially, GeoRank does not care about the total number of source users followed by the querying user, it will always process up to k source users for the news feed. As a result, we can infer that GeoRank will be even more efficient comparing with the *on-top* approach for the active users, who may follow hundreds or thousands of other users in a real social networking system.

Based on the above experiments, GeoRank outperforms the *on-top* approach significantly in all the cases. Thus, we can safely conclude that GeoRank will reach a better throughput than the *on-top* approach. Moreover, we find that the *on-top* approach is impractical due to its unacceptable performance (i.e., the response times), and hence will not consider it in the further experiments.

4.6.2 GeoRank Query Processor Performance

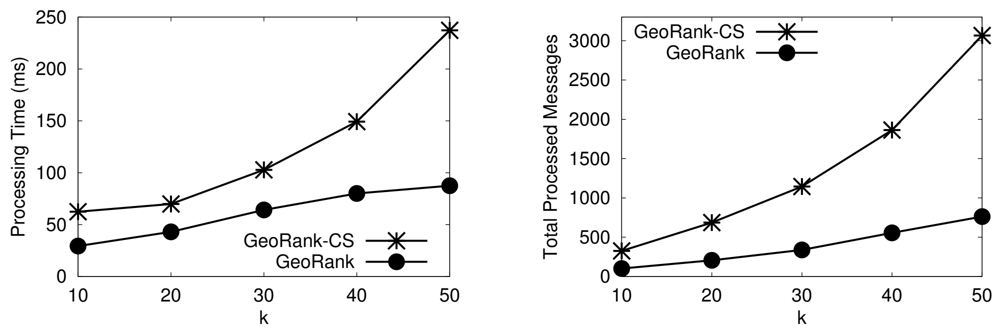
This section studies the performance of the internals of GeoRank *query processor*, i.e., the processing time to produce the news feed. As discussed in Section 4.4, GeoRank *query processor* has two main steps: *candidate sources selection* and *news feed aggregation*. To study the effect of each step separately, we compare two versions of GeoRank. The first version, termed *GeoRank-CS* applies only the *candidate sources selection*, while the second one is equipped with both the pruning steps. The comparison is done when varying the number of source users, k, and ω.

Number of source users. Figure 4.5 gives the performance of both GeoRank and GeoRank-CS when varying the number of source users from 100 to 300. The performance is measured in terms of processing time (Figure 4.5a) and number of processed messages (Figure 4.5b). The processing time for both GeoRank and GeoRank-CS increases with the number of sources, as we need more time to calculate and rank the scores of most relevant messages in the first step.



(a) Processing Time. (b) Total Processed Messages.

Figure 4.5: Different Numbers of Source Users

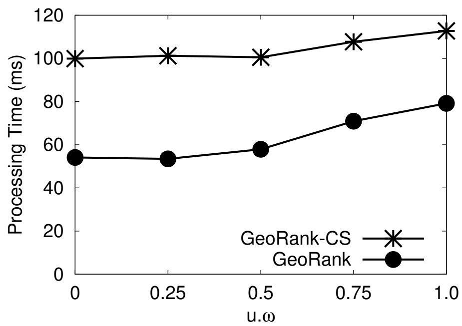


(a) Processing Time. (b) Total Processed Messages.

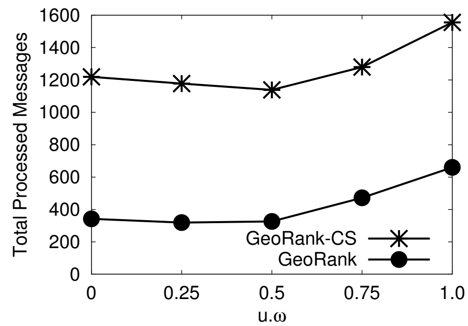
Figure 4.6: Different Numbers of K Values.

The performance gap between GeoRank and GeoRank-CS shows the effect of the *news feed aggregation* step, employed by GeoRank, where it: (a) prunes more sources than GeoRank-CS with the early termination condition, and (b) employs both spatial and temporal pruning techniques to avoid retrieving k messages from each candidate source. With number of processed messages (Figure 4.5b), it is interesting to see a consistent behavior for both GeoRank-CS and GeoRank, as the number of processed message is almost not affected by the number of sources. This is mainly as in both algorithms, the *candidate selection step* prunes the list of sources to 30, i.e, the value of k. As a result, no matter how many sources we have, we only operate on the top-30 of them.

Top k values. Figure 4.6 gives the performance of both GeoRank and GeoRank-CS, when varying k from 10 to 50, in terms of processing time (Figure 4.6a) and number of processed



(a) Processing Time

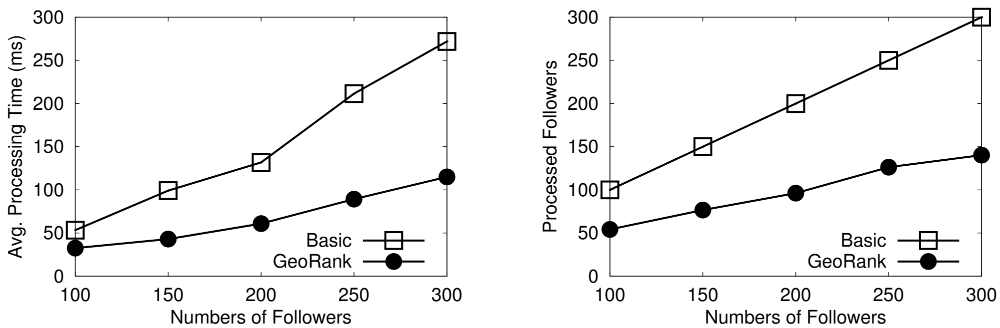


(b) Total. Processed Messages.

Figure 4.7: Different User Preferences.

messages (Figure 4.6b). With the increase of k, Both GeoRank and GeoRank-CS take more time to produce the answer and process more messages, as both of them need to check more sources and messages. However, it is interesting to see that the performance gain of GeoRank increases with k. This shows that with larger k, the *news feed aggregation* step in GeoRank becomes even more effective, because of the pruning and early termination. Also, it is obvious that the trend of processing time is similar to the trend of the number of processed messages, which gives insight that the processing overhead is mainly due to the processed messages.

User preference parameter u.ω. Figure 4.7 gives the performance of both GeoRank and GeoRank-CS when varying ω from 0 (spatial proximity is mostly favored) to 1 (temporal proximity is mostly favored), in terms of processing time (Figure 4.7a) and number of processed messages (Figure 4.7b). For all values of ω, there is a consistent performance gap between GeoRank and GeoRank-CS. Although ω plays a major role in determining both the spatial and temporal boundaries in the *news feed aggregation* step, increasing or decreasing the value of ω just tightens one boundary and relaxes the other. Overall, the performance gain becomes consistent. In the mean time, for $\omega > 0.5$, we can see that the number of processed messages increases for both GeoRank and GeoRank-CS, hence, the performance degrades. This is mainly due to the distribution of messages is more clustered towards the time domain, and hence favoring the temporal domain results in processing more messages.



(a) Processing Time.

(b) Processed Followers.

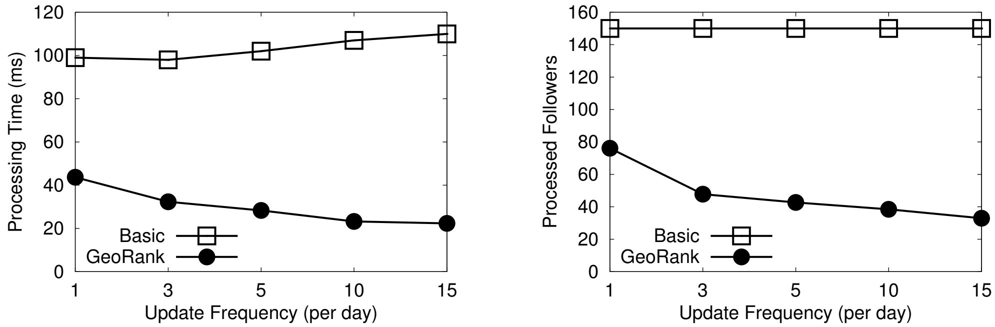
Figure 4.8: Different Numbers of Followers.

4.6.3 GeoRank Message Updater Performance

This section studies the performance of GeoRank *message updater* module, which is the overhead to maintain the statistics. As discussed in Section 4.5, GeoRank *message updater* employs a set of monitoring areas, indexed by a spatial grid index, to determine which followers will be affected by the new message. To evaluate the main idea of the *message updater* module, we compare the full version of GeoRank against a *basic* approach for the *message updater* module, which scans all the followers for most relevant message updates whenever a new message is posted. We discuss the results varying the number of followers, update frequencies, and ω.

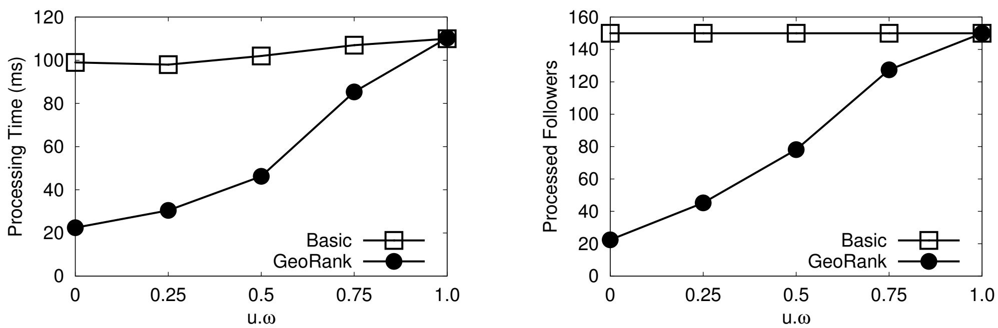
Number of followers. Figure 4.8 gives the performance of both GeoRank and *basic* approaches when varying the number of followers $|u.\mathcal{F}|$ from 100 to 300. In terms of average processing time for each posted message (Figure 4.8a), both GeoRank and *basic* encounter more time with the increase of the number of followers. However, *basic* approach uses more time with more followers, as it needs to check all the followers. On the other side, GeoRank is more efficient, as it utilizes the monitoring areas and only updates a subset of followers, as we plot the number of followers (Figure 4.8b). For example, in the case of 300 followers, the *basic* approach checks on 300 followers while GeoRank checks on only 140 followers.

Message update frequencies. Figure 4.9 gives the performance of both GeoRank and *basic* approaches when varying the message update frequency from 1 to 15 messages per day. In terms of average processing time for each posted message (Figure 4.9a), the *basic* approach suffers from an increase in processing time with higher frequencies, where we need to check on every follower for update. In the mean time, GeoRank shows much better scalability where



(a) Processing Time. (b) Processed Followers.

Figure 4.9: Different Message Update Frequencies.



(a) Processing Time. (b) Processed Followers.

Figure 4.10: Different User Preference Parameters.

it actually gives better performance with higher update frequency. The main reason behind this scalability is that with higher update frequencies, GeoRank can easily update the monitoring areas for the user followers to be tighter and more accurate. Thus, higher update frequency results in a much better performance for GeoRank. In terms of the number of followers to check on (Figure 4.9b), the *basic* approach gives steady state performance as it basically checks for all the 150 followers. In the mean time, GeoRank checks for lower number of followers with the increase of the update frequency. This is mainly due to the tighter and more accurately calculated monitoring areas, as previously discussed.

User preference parameter. Figure 4.10 gives the impact of different follower's preference parameter $u.\omega$ on the message update performance of GeoRank and *basic* approach, where we set the same preference parameter for all the user's followers varying from 0 (only cares about location) to 1 (only cares about time). Figure 4.10a gives the average processing time, and Figure 4.10b gives the average number of followers processed by each approach for one message update. We have the following observations: (1) the basic approach is almost not affected by the ω at all, as it checks the most relevant message update for all the followers, regardless of their preference parameter ω, (2) GeoRank is more scalable than the *basic* approach as it only updates a subset of the followers for each message update, and (3) GeoRank has a better performance, when the follower's preference parameter is smaller (cares more of the spatial domain). This is because the monitoring areas are calculated based on Equation 4.4, where with a larger preference parameter $u_f.\omega$, we will have a smaller monitoring area registered at the user's spatial index. As a result, less number of followers is selected for most relevant message updates, when a new message is posted from the user. On the other hand, when the follower's preference parameter is larger (i.e., cares more about time), the processing time of GeoRank increases, as each follower has a larger monitoring area and more followers are selected for update when a new message is posted. For example, in the extreme case $u_f.\omega = 1$ (meaning that the followers only cares the most recent message from the source users), GeoRank has the same performance as the basic approach, because, in this case, the monitoring area is set as the whole space, and all the followers need to be selected for the most relevant message update.

4.7 Summary

In this chapter, we have presented GeoRank; an efficient location-aware news ranking system. GeoRank provides the top-k relevant news items considering: (a) the spatial proximity, (b) the time recency of the message, and (c) a preference function, that weights the relative importance of the above factors. The basic ideas in GeoRank is fairly simple but highly effective, which tries to avoid the unnecessary computations for processing the disqualified the news sources and their messages. GeoRank is composed of two main modules, namely, *query processor* and *message updater*. The *query processor* module retrieves top-k most relevant news feed. On the other side, the *message updater* module is a process running in the background to maintain the statistics. Such statistics ensure efficiency of the *query processor*, and hence GeoRank. Extensive experimental results, based on real and synthetic data sets, confirm that GeoRank significantly reduces the response time for news feed and improves the efficiency by at least 6 to 10 times.

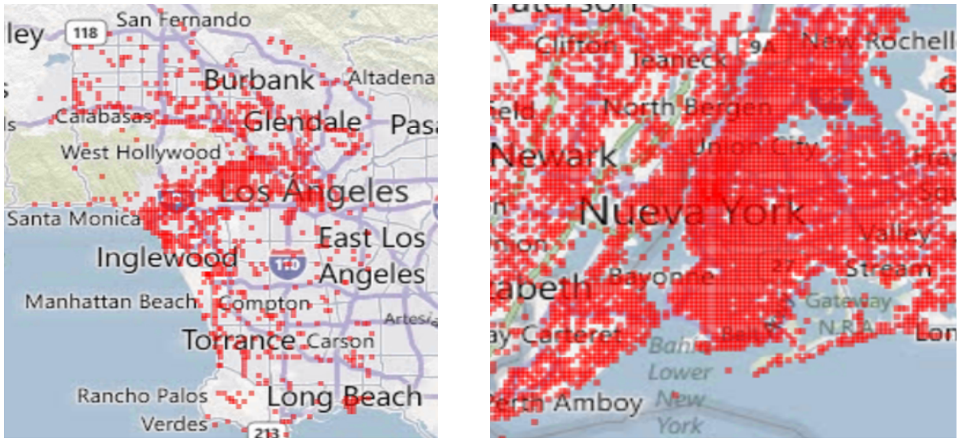
Chapter 5

Preference-Aware Location-based Recommendations

5.1 Introduction

The advances in location-acquisition and wireless communication technologies enable people to add a location dimension to traditional social networks, fostering a bunch of location-based social networking services (or LBSNs) [83], e.g., Foursquare, Loopt, and GeoLife [84], where users can easily share life experiences in the physical world via mobile devices. For example, a user can leave comments with respect to a restaurant in a LBSN site, so that the people from her social structure can refer to the comments when they visit the restaurant in a later time. Location as one of the most important components of user context implies extensive knowledge about an individual's interests and behavior, thereby providing us with opportunities to better understand users in a social structure according to not only online user behavior but also the user mobility and activities in the physical world. For instance, people often visiting gyms might like physical exercises and users who usually have dinner in the same restaurant may share a similar taste. Sometimes, individuals who do not have overlaps of physical locations can still be linked, as long as the categories of their visited locations are indicative of a similar interest, such as beaches or museums.

Under such a circumstance, a location recommender system is a valuable but unique application in location-based social networking services, in terms of what a recommendation is



(a) New York users in Los Angels (b) New York users in New York City.

Figure 5.1: User Location History Distributions.

and where a recommendation is to be made [85, 83]. Specifically, location recommendations provide a user with some venues (e.g., an Italian restaurant or a fancy movie theater) that match her personal interests within a geospatial [83]. This application becomes more worthy when people travel to an unfamiliar area, where they have little knowledge about the neighborhoods. Nevertheless, a high-quality location recommendation has to simultaneously consider the following three factors. 1) *User preferences*: For example, food hunters maybe more interested in the high quality restaurants, while the shoppingaholics would pay more attentions to nearby shopping malls [86]. 2) *The current location of a user*: As the users prefer the nearby locations, this location indicates the spatial range of the recommended venues and may affect the ratings of these recommendations [12]. 3) *The opinions of a location given by the other users*: Social opinions from the nearby users is a valuable resource for making a recommendation [87]. But, the most popular venue may not always fit a particular user given her distinct preferences.

Inferring the rating for a location is very challenging using a user's location history in a LBSN. First, a user can only visit a limited number of physical locations. This results in a sparse user-location matrix for most existing location recommendation systems, e.g., [12, 87], which directly play a collaborative filtering-based model [88, 89] over physical locations. Second, the task becomes even more difficult when an individual travels to a new place where she has visited few locations (though we believe people need the location recommendation service most at this moment). For example, Figure 1 a) and b) plot the locations (according to the *tips* in Foursquare) visited by people from New York City, in Los Angles (LA) and New York City (NYC) respectively. Clearly, the *tip* records generated by NYC people are very few in LA, which are only 0.47% of the records they left in NYC and 0.75% of the records generated by local users in LA. This phenomenon is quite common in the real world [90], aggravating the data sparse problem to location rating inference (if we want to provide people from NYC with location recommendations in LA). In this case, solely using a CF model is not feasible any more. First, we cannot simply put together the location histories of users from different cities in to a user-location matrix, which is neither efficient nor scalable. Second, performing collaborative inference in each city separately cannot cope with the *new city* problem demonstrated in Figure 5.1 a) very well, as a user usually has not enough location history in a city that is new to her.

To this end, we report on a location-based and preference-aware recommender system that offers a particular user a set of venues (such as restaurants and shopping malls) within a user specified geospatial range with the consideration of the three factors mentioned in the third paragraph. By modeling a user's preferences based on the category information of her location history (instead of physical locations) in a LBSN, our recommender system can facilitate people's travel not only near their living areas but also to a city that is new to them. Generating such a location recommendation is challenging because of two reasons:

1) *Learning a user's preferences.* First of all, a user's preferences are usually comprised of multiple kinds of interests, such as shopping, watching movies, cycling, and arts. By the meantime, a user's preferences are not generally binary decisions, e.g., like or dislike something, and have a variety of granularities, such as "*Food* $\rightarrow$ *Italian food* $\rightarrow$ *Italian noodles*". In addition, a user's preferences are evolving from time to time. Manually specifying an individual's preferences with some words is impractical. As a result, unobtrusively modeling a user's preferences with her location history is non-trivial.

2) *Inferring the rating to an unvisited location for an individual.* The rating inference needs to consider both an individual's preferences, the opinions given by the other users, especially the *local experts* [91, 92], and the similarity between them. This inference demands three aspects of computing: a) estimating the expertise of a user, b) computing the similarity between users, and c) collaborative social opinion inference for a location incorporating the results of the former two computation, e.g., using collaborative filtering (CF) model [88, 89]. None of them are

trivial.

Specifically, our contributions can be summarized as:

• We learn a user's preferences from her location history and model the preferences with a weighted category hierarchy (WCH). We further estimate the similarity between two users' preferences by computing the similarity between the two users' WCHs. This method contributes to user preference modeling and handling the data sparseness problem for location recommendations.

• We pre-compute and extract the *local expert* for each location category in a city using an iterative inference model over the users' location histories there, which improves the efficiency of our online recommendation process.

• We online infer the rating to a venue with the *local experts* selected by a preference-aware candidate selection algorithm and a CF-based model. This approach enables a real-time location recommendation simultaneously considering an individual's location, preferences granularities, and opinions from *local experts*.

• We evaluated our system with a real-world dataset collected from Foursquare including 221,128 tips generated by 49,062 users in NYC and 104,478 tips generated by 31,544 users in LA. The extensive experimental results show that our method provide users with location recommendations more effectively and efficiently beyond the existing baselines.

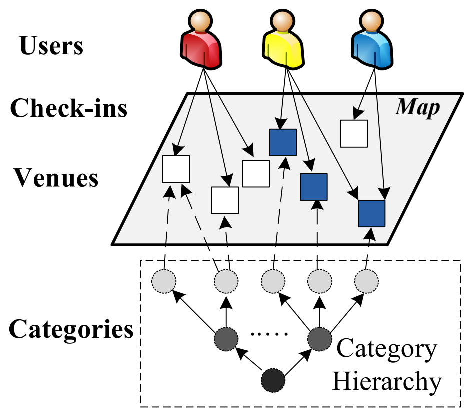
The rest of the chapter is organized as follows: Section 5.2 gives an overview of our system. Section 5.3 and Section 5.4 present two major parts of our system: 1) *offline modeling* and 2) *online recommendation*. Extensive experimental results based on the real dataset are provided in Section 5.5 with some discussions. Finally, Section 5.6 concludes the paper.

5.2 System Overview

This section first introduces the key data structures we will use in the paper, and then presents the application scenario and overall architecture of the proposed location recommender system.

5.2.1 Preliminary

Figure 5.2 illustrates the relations of five key data structures: 1) *user*, 2) *venue*, 3) *check-in*, 4) *user location history* and 5) *category hierarchy*. In a location-based social network, a user u maintains her profile information, such as ID, name, age, gender, and home town. Moreover,



(a) Overview of a location-based social network

Category Name	Number of sub-categories
Arts & Entertainment	17
College & University	23
Food	78
Great Outdoors	28
Home, Work, Other	15
Nightlife Spot	20
Shop	45
Travel Spot	14

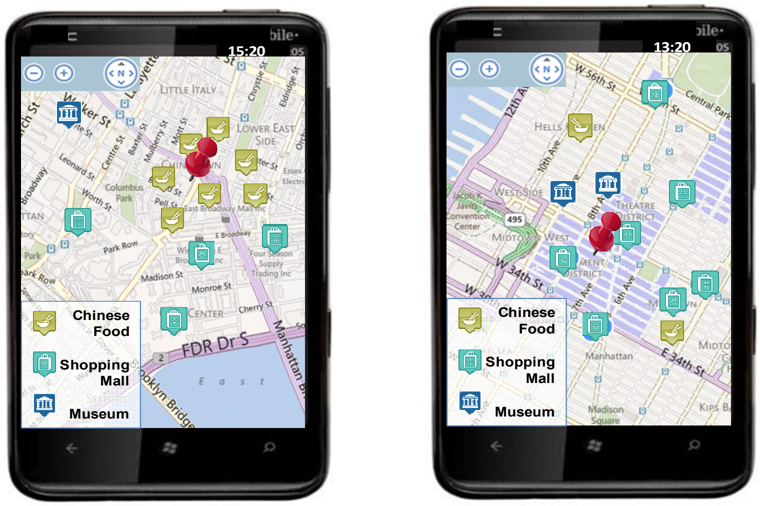
(b) Detailed location category hierarchy in FourSquare

Figure 5.2: Data Structures in Location-Based Social Networks.

the user can also mark a venue (e.g., a restaurant) and leave some comments, when she arrives there, which is also known as *check-in* in a LBSN. A user can visit multiple locations and may generate a *check-in* for each of the visit, shown as the solid arrows in Figure 5.2 a). All of the user's *check-ins* reflect her *location history* in the real world. Depicted as squares on the map, a venue is a location associated with a pair of coordinates indicating its geographical position and a set of categories denoting its functionalities. The categories of venues have different granularities, which are usually represented by a *category hierarchy* shown in the bottom part of Figure 5.2 a). For example, "Food" category includes "Chinese restaurant" and "Italian restaurant" and etc. In our system, we focus on a two-level category hierarchy obtained from Foursquare, as shown in Figure 5.2 b).

5.2.2 Application Scenario

Figure 5.3 demonstrates an application scenario of our system, where the top N (N=10 here) venues matching a user's preferences are recommended based on the geo-region of the present view. Here, the number of recommendations and scale of the geo-region are determined by a user (e.g., by zooming in/out and panning a map in Figure 5.3, while the ranking of the locations are calculated in our backend system, based on the location history of the user and



(a) Recommendations near Chinatown

(b) Recommendations near 7th Ave

Figure 5.3: Example of An Application Scenario in NYC.

the opinions from the other people. Generally, the number of locations belonging to a category in the recommendations follows the distribution of the categories in the user's preferences. For example, the user (whose location is represented by the push-pin in Figure 5.3) has "Chinese restaurants" as her most preferred location category and "Shopping malls" as the second. Then, as demonstrated in Figure 5.3 a), "Chinese restaurants" have the biggest presence and shopping malls are the second in the recommendations, when she is near the Chinatown. However, when we change the map view to the 7th Ave, as shown in Figure 5.3 b), the presence of malls could become the majority of the recommendations though Chinese restaurants is her first interest. The reason is that the malls have much higher quality than the Chinese restaurants, according to people's location histories in that particular area. This is a trade-off between the user preferences and social opinions.

5.2.3 System Architecture

Offline Modeling. The offline modeling part is comprised of two major components: 1) *social knowledge learning* and 2) *personal preference discovery*, as illustrated in the lower half of Figure 5.4. The first component infers each user's expertise in each category city-by-city

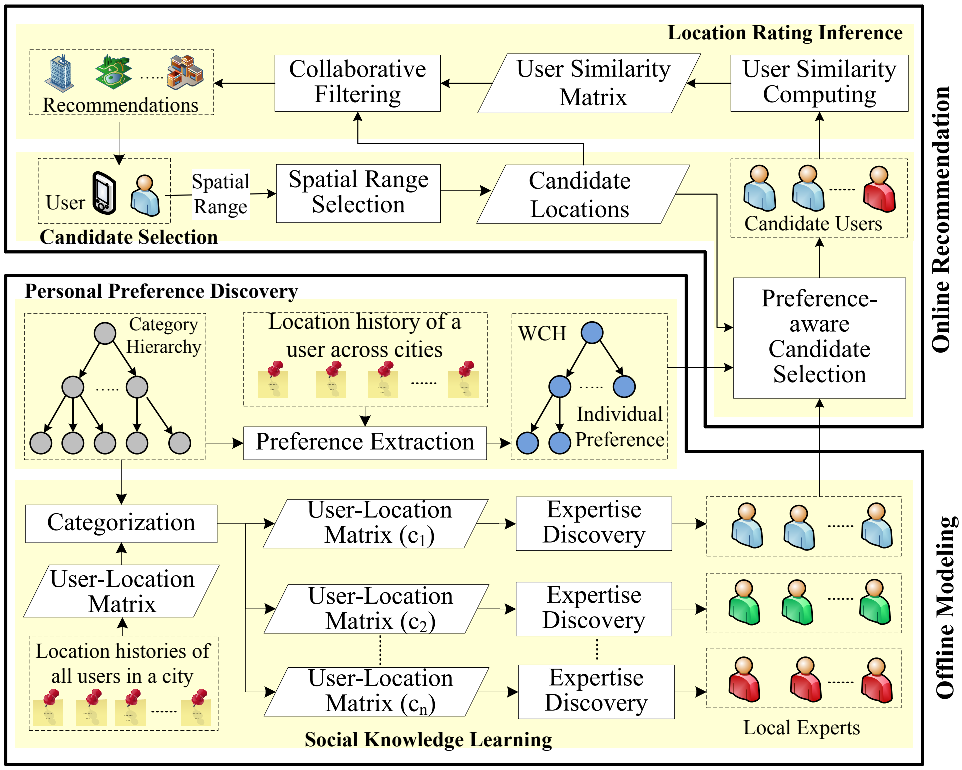


Figure 5.4: System Architecture.

according to their location histories. Given a pre-defined category hierarchy (e.g., Figure 5.2 b), we break a user's location history in a city into groups of different location categories. Then, we model each category group of location histories using a user-location matrix, in which each entry denotes a user's number of visits to a physical location. By applying an iterative inference model to each user-location matrices, we calculate a score w.r.t. a category for each user, indicating a user's expertise in that category in that city. By ranking the users in terms of the score corresponding to a category, we can discover the *local experts* of different categories in the city. The inferred expertise of a user will be used in later preference-aware candidate selection algorithm and help the online part generate quality recommendations with fewer computational loads. The second component models each user's personal preferences using a WCH by taking advantage of the location category information lying her location history, which help us to overcome the data sparsity problem. Specifically, a WCH is a sub-tree of the predefined category hierarchy, where each node carries a value denoting the user's number of visits to a category. These values are further normalized on each layer of a WCH using TF-IDF (term frequency-inverse document frequency) [93].

Online Recommendation. The online recommendation part provides a user with a list of venues, considering the user's preferences, current location, and social opinions from the selected *local experts*, detailed in the following two components: 1) *Preference-aware candidate selection.* This component selects a set of *local experts* who visited the venues within a user's recommendation range R and have a high expertise in the categories preferred by the user. A preference-aware candidate selection algorithm is designed to properly choose these *local experts* from different categories according to a user's different preference weights in her WCH. Meanwhile, this algorithm improves the efficiency of our approach significantly while maintaining the effectiveness, making our system really location-aware. 2) *Location rating calculation.* This component first computes the similarity between each selected *local expert* and the user using a similarity function based on their WCHs. The calculated similarity score is further fed into a CF-based model to infer the rating that the user would give to an unvisited candidate venue. Later, the venues with relative high predict ratings are returned as the location recommendations.

5.3 Offline Modeling

In this section, we present the *offline modeling* part of our system, which is comprised of: 1) *Social knowledge learning*, which evaluates a user's experiences and discovers the *local experts* in each city, and 2) *Personal preference discovery*, which extracts a user's preferences from her location history.

5.3.1 Social Knowledge Learning

To identify the *local experts* of a location category like "Chinese food" and "shopping mall", this component computes a user's expertise in each category in different cities based on category information encapsulated in the user's location history. Intuitively, *local experts* of a category can find high quality venues of the category as compared with the regular users, resulting in more valuable location histories for a reference. In addition, using the *local experts* we are able to ignore some random users who have little data (and knowledge) in a category of locations,

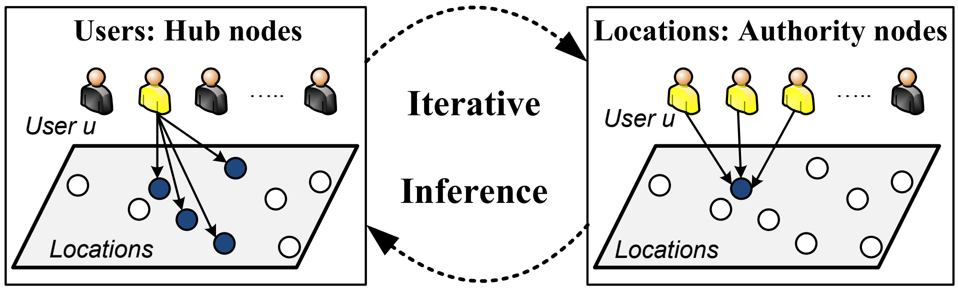


Figure 5.5: The Iterative Model for Social Knowledge Learning.

thereby reducing unnecessary computation during the online recommendation.

In our method, we first partition all users' location histories by cities as a user's knowledge usually varies in terms of geographic spaces, e.g., a travel expert of New York City may have no idea about the interesting venues in Beijing. Moreover, users may have different expertise in different location categories, e.g., a user likes "Chinese food" in the city does not necessary have much knowledge about "Italian food" there. Thus, we further divide users' location histories in a city into groups according to the categories of their visited venues. As a result, a city has n user-location matrices (n is the number of predefined categories) where an entry denotes the number of visits of a user to a venue. Later, we apply a HITS (or Hypertext Induced Topic Search)-based inference model [94, 95] to each category-based user-location matrix, inferring the expertise of each user in that category. As shown in Figure 5.5, this model regards an individual's visit to a venue as a directed link from the user to that venue. Each user has a hub score denoting its knowledge and each location is associated with an authority score indicating its interest level. The insight supporting this model is the mutual reinforcement relationship between a user's knowledge and the interest level of a venue [62]. That is, people who have visited many high quality venues in a region are more likely to have rich knowledge about that region. In turn, a venue visited by many people with rich knowledge is more likely to be a quality venue. As a result, as shown in Equation 5.1 and 2, a user's knowledge can be represented by the sum of the authority scores (i.e., interest levels) of the venues visited by the user, and the interest level of a venue can be represented by the sum of the hub scores (or knowledge) of the users who have visited this venue. Using a powerful iteration inference method, we generate the final scores for each user and each venue. The users with a relatively high authority score are regarded as the *local experts* in that category.

$$v_c.a = \sum_{u \in \mathcal{U}} u_c.h \tag{5.1}$$

$$u_c.h = \sum_{u.v \in c} v_c.a \tag{5.2}$$

where $u_c.h$ is user u's hub score in category c and $v_c.a$ denotes venue v's authority score.

If we use $\mathcal{A}_n$ and $\mathcal{H}_n$ to denote authority and hub scores at the nth iteration and $\boldsymbol{M}$ as the user-category matrix, the iterative processes for generating the final results are:

$$\mathcal{A}_n = \boldsymbol{M}^T \cdot \boldsymbol{M} \cdot \mathcal{A}_{n-1} \tag{5.3}$$

$$\mathcal{H}_n = \boldsymbol{M} \cdot \boldsymbol{M}^T \cdot \mathcal{H}_{n-1}, \tag{5.4}$$

as we set the initial authority and hub scores as the number of a user's visits, we are able to calculate the authority and hub scores using the power iteration method and identify the *local experts*.

5.3.2 Personal Preference Discovery

We extract a user's preferences from the category of her visited locations. As illustrated in Figure 5.6, we first project a user's location history across all the cities onto a predefined category hierarchy, where nodes occurring on a deeper layer denote the categories of a finer granularity. As a result, each node is associated with a value representing the number of visits (of the user) to a category. This is motivated by the fact that an individual's preferences are usually made up of multiple interests (such as shopping and hiking), which further have different granularities, e.g., "Food" $\rightarrow$ "Chinese food". Second, we calculate the TF-IDF value of each node in the hierarchy, where a user's location history is regarded as a document and categories are considered as terms in the document. Intuitively, a user would visit more locations belonging to a category if the user likes it. Further, if a user visits locations of a category that is rarely visited by other people, the user could like this category more prominently. For example, the number of visits to restaurants is generally more than other categories like museums in people location histories. It does not mean food is the first interest of all the people. However, if we find a user visits museums very frequently, the user may be truly interested in arts or history.

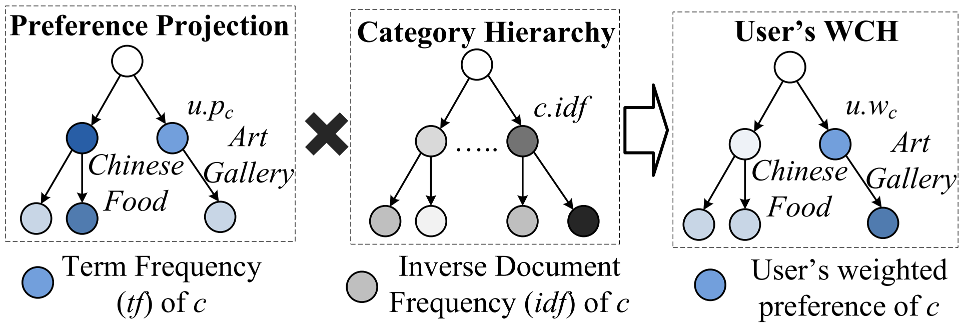


Figure 5.6: User WCH Construction.

Overall, a user's preference weight ($u.w_{c'}$) is calculated by Equation 5.5, where the first part of the equation is the TF value of category c in user u's location history and the second part denotes the IDF value of the category.

$$u.w_{c'} = \frac{|\{u.v_i : v_i.c = c'\}|}{|u.\mathcal{V}|} \times \lg \frac{|\mathcal{U}|}{|\{u_j{:}c' \in u_j.\mathcal{C}\}|}, \tag{5.5}$$

where $|\{u.v_i : v_i.c = c'\}|$ is user u's number of visits in category c', $u.\mathcal{V}$ is the total number of the user's visits, and $|\{u_j{:}c' \in u_j.\mathcal{C}\}|$ counts the number of users who have visited category c' among all the users $\mathcal{U}$ in the system. Clearly, after applying IDF to the user's WCH, Chinese restaurant is no longer the first preference (i.e., with lighter color). The WCH well captures a user's interests, having the following advantages: 1) reduce the concern raised by the different data scales of different users, 2) handle the data sparseness problem and reduce the computational loads for further user similarity computing (from physical locations to categories), and 3) enable the computing of similarity between users who do not share any physical location histories, e.g., living in different cities.

Algorithm 1: Preference Aware Candidate Selection

Input: (1) Spatial Region R, (2) A user's *u.wch*, and (3) Total number of location recommendations N.
Output: (1) A set of selected local experts E and (2) A set of candidate locations V

```
Retrieve venues V' in R
U ← users who have visited V'
while True do
    for level l from bottom to the root-1 in u.wch do
        w_min ← minimum preference weight at l
        for each category c in user's u.wch at level l do
            k ← |u.w_c/w_min|        //Calculate the number of users
            e ←Top(k, U, c)          // Select top-k users based on u'_c.h
            for each u' ∈ e do
                V ← V ∪ u'.V located in R
            E ← E ∪ e
        if enough candidate venues |V| ≥ N or E == U then
            Return local experts E and candidate locations V
```

5.4 Online Recommendation

In this section, we present *online recommendation* part of our system, which consists of: 1) *preference-aware candidate selection,* which selects the candidate*local expert* based on the user's preferences and 2) *location rating calculation*, which infers a predication score of the candidate locations the user would give based on CF-based inference model using the similarity comparison between the user and selected *local experts*.

5.4.1 Preference-Aware Candidate Selection

This component selects a set of candidate *local experts* and venues in the user specified geospatial range using our preference-aware candidate selection algorithm (i.e., demonstrated as Algorithm 1), which guarantees the number of selected venues exceeds the individual's requirement k and the category distribution of the selected *local experts* fits the individual's preferences. The algorithm significantly improves the efficiency of the online recommendations process as

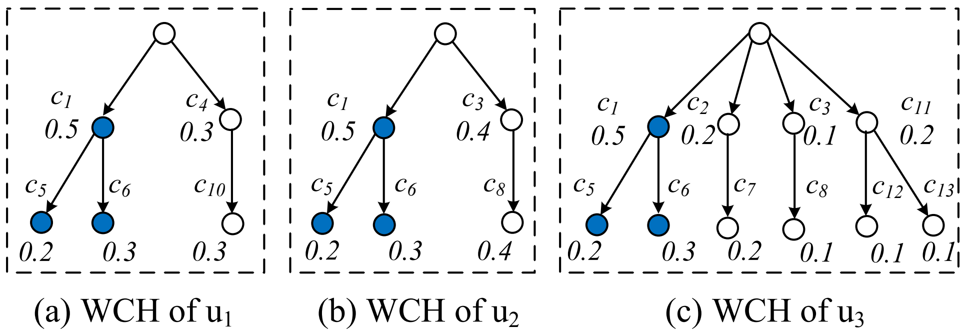


Figure 5.7: Diversities of Users' Preferences.

we do not need to compute the similarity between the individual and all the users in the area any more. Meanwhile, the location history of users with very little knowledge about the region can be excluded, as they may have limited contributions to the final score inference. The experiments show that the candidate selection increases the efficiency significantly while maintaining the effectiveness.

Specifically, given a geospatial range R specified by the individual, this algorithm first retrieves the venues V' located in the range and users U who have visited these venues (Line 1 and 2).The candidate *local experts* selection process initiates from the bottom level of the individual's WCH (which has a finer granularity) and moves up to the next higher level if the number of venues cannot meet the required number of recommendations. When selecting venues at one level of WCH, we choose the node (a category) having the minimum value w_{min}. Later, we calculate a k value using $\lfloor \frac{u.wc}{w_{min}} \rfloor$ to decide the number of *local experts* we select in this category, and then top-k users with a relatively high expertise (hub score) in category c are selected as candidate experts e (Line 7-8). The venues (located in R) visited by the users in e will be retrieved and deposited into V. After that, candidate experts e are merged with E (Line 9-11). The algorithm will stop once we obtain enough number of venues or all the users who have visited region R have been scanned. As a result, a set of venues V and a set of *local experts* E are returned.

5.4.2 Location Rating Inference

Step 1. User Similarity Computing. In this step, we compute a similarity score between an individual (who issues the recommendation request) and each *local expert* (selected by Algorithm 1) according to their WCHs. Since a WCH is essentially a tree, we measure the similarity between the two WCHs in terms of both their structures and the preference weights associated with each overlapped node. Specifically, we decompose the similarity between two WCHs as a weighted sum of the similarities between each corresponding level of the WCHs (i.e., $u.wch.l_1$ vs. $u'.wch.l_1$). The deeper levels are given a bigger weight as they represent a finer granularity of an individual's preferences. Further, the similarity between the same levels of two different WCHs is measured by the following two aspects:

The first one is the number of overlapped nodes at the level and their values, as shown in Equation 5.6. The more overlapped nodes two WCHs have the more similar the two users could be. The minimum preference weight of an overlapped node c is selected to represent two users' common interests.

$$\textit{LevelSim}(u, u', l) = \sum_{c \in \mathcal{C}^l} \textit{min}(u.w_c, u'.w_c), \tag{5.6}$$

The other is the entropy of each level, which can effectively capture the diversity of a user's preferences [96], as shown in Equation 5.7, where $H(u, l)$ is user u's entropy at level l and $P(c)$ is the probability that u visited category c in her historical data.

$$H(u, l) = -\sum_{c \in \mathcal{C}^l} u.P(c) \times \lg u.P(c), \tag{5.7}$$

Figure 5.7 illustrates the importance of this entropy using an example, where three users share some same preferences (marked blue in WCHs) and the values represent the weights. Without considering the entropy of each level, the similarity scores *Sim*(u_1, u_2) and *Sim*(u_1, u_3) are identical. However, we can clearly observe that u_1 is more similar to u_2 who is relatively focused than u_3 who has a variety of interests. Or, we can say u_3 is more different from u_1 as compared with u_2 since u_3 has more different categories. We validated the effectiveness of the entropy in later experiments.

Finally, the similarity between two WCHs can be calculated as Equation 5.8, where β is a weight varying in the depth of the level of the location category (the depth of a root is 0) in the hierarchy. In the experiment we choose β=2^l as we found the overlapped nodes decreased exponentially as the depth of levels increases.

$$Sim(u,u') = \sum_{l=1}^{|l|} \beta \times \frac{LevelSim(u,u',l)}{1+|H(u,l)-H(u',l)|} \tag{5.8}$$

That is, two users are more likely to be similar if 1) they share more nodes with a bigger preference weight, 2) the difference between each level's entropy is small, and 3) these nodes located in a lower level in their own WCHs.

Step 2. Location Rating Calculation. In this step, we place the *local experts* and candidate venues selected by Algorithm 1 back into a user-location matrix, which is fed into a user-based CF model to infer a user's rating of a candidate venue. The general intuition behind a CF model is that similar users rate the same items similarly. As users usually do not offer explicit ratings to a venue in a LBSN, we regard a user's number of visits to the venue as an implicit rating (of the venue). Formally, the rating that user u would give to venue v is calculated as Equation 5.9.

$$R_u(v) = \sum_{u' \in \mathcal{E} \& v \in \mathcal{V}} Sim(u,u') \times v(u',v), \tag{5.9}$$

where $v(u',v)$ denotes the number of visits of user u' at venue v. Note that the user similarity $Sim(u,u')$ is computed in the Step 1 based on WCHs rather than the simple Cosine similarity between two users's location vectors. That is, we can still make recommendations for a user even if the user has not visited any locations in a new city. Finally, the system returns the top-N venues with the highest scores to the user as the location recommendations.

5.5 Experimental Evaluation

In this section, we first describe the settings of experiments including the dataset, baseline approaches, and the evaluation method. After that, we report on major results on both the effectiveness and efficiency of our system followed by some discussions.

5.5.1 Experiment Settings

Datasets. We study the top two largest cities in USA, obtaining 221,128 *tips* generated by 49,062 users in New York City (NYC) and 104,478 *tips* generated by 31,544 users in Los

Angeles (LA) from Foursquare. At the meantime, we collect these users' *tips* in other cities so as to model a user's preferences thoroughly. Foursquare blocked the API for crawling a user's *check-in* data due to the privacy concern, but leaving tips open to download. Our method could be more effective if using *check-in* data (though it is not bad using the *tips*). On the other hand, *tips* have their own advantages in reflecting a user's real interests. Some-times, people check in at a venue without doing anything at the venue. But, leaving a *tip* in a venue usually means a user has carried out some essential activities (like dinning and shopping) at the venue.

The following information is recorded when collecting the data: 1) user profile information, including the user ID, name, and home city; 2) venue profile information, consisting of a venue's ID, name, address, GPS coordinates, and its categories; and 3) user location histories, represented by all the *tips* a user left in the system. Each *tip* is associated with a venue ID, comments and a timestamp. From the dataset we collected, we choose the users whose home city is located in New Jersey (NJ) state and study the location recommendations made for these users in NYC and LA respectively. To guarantee the validity of the experimental results, we further select the user who has over 8 *tips* in a city as a candidate query user. Table 5.1 shows the details about these NJ users, where the footprint range denotes the average diagonal distance of the minimal bounding box of the locations visited by the user in the querying city. The data presented in Table 5.1 tells two stories. First, users have more opportunities traveling to nearby locations, thereby generating more *tips* in total in a nearby city than a distant one. Second, users who visit LA traveled in a large range than those visiting NYC. This is in line with the fact that LA is larger than NYC geographically.

Baseline approaches. We compare our method with the following three baseline approaches, detailed in Table 5.2, where the first three baseline approaches are the existing recommender systems and the fourth one (ours w/o CS) means our method without using the preference-aware candidate selection algorithm.

Home City	**Querying City**	**Total Users**	**Tips in City**	**Tips /User**	**Footprint (miles)**	**All Tips**
NJ	LA	228	2,553	11.20	5.31	9,836
NJ	NYC	2,886	72,170	25.01	3.93	106,870

Table 5.1: Statistics of Experimental Data Set.

1) *Most-Preferred-Category-based (MPC) recommendation.* Given a user-specified geospatial range and the user's WCH, this approach chooses the top-N venues as the final recommendations based on an iterative inference model, which is similar to [62]. As compared with our method, this approach does not consider local users' opinions on the recommended locations.

2) *Location-based Collaborative Filtering (LCF).* Location-based Collaborative Filtering (LCF) is the most common way that people would come up with [39], which applies the collaborative filtering method directly over the venues. This baseline utilizes the users' location histories in a city with a user-venue matrix (an entry denotes the number of visits of a user to a venue) and applies the traditional user-based CF method to make recommendations. The Cosine similarity between two users' location vector is employed as the similarity between the two users, and the inference is performed offline. Finally, the locations in the user-specified range and having a relatively inference score will be recommended.

3) *Preference-based Collaborative Filtering (PCF).* This baseline first retrieves all the users and venues in the user-specified range, formulates a user-venue matrix online, and then applies a user-based CF model to predict a user's rating of a venue. This approach starts considering the opinions from other users. However, the similarity between two users is represented by the Cosine similarity between the category vectors corresponding to the two users (without considering the category hierarchy).

Method	**Social Opinion**	**Category of Location**	**Preference Hierarchy**	**Candidate Selection**
MPC		$\surd$	$\surd$	$\surd$
LCF	$\surd$			
PCF	$\surd$	$\surd$		
Ours w/o CS	$\surd$	$\surd$	$\surd$	
Ours	$\surd$	$\surd$	$\surd$	$\surd$

Table 5.2: Comparison Between Baseline Methods and Ours.

Evaluation methods. We evaluate both the effectiveness of the suggested recommendations and the efficiency for generating online recommendations with the baseline solutions.

1) *Recommendation effectiveness.* It is very difficult to carry out a large-scale in-the-field study for evaluating the effectiveness of the location recommendations. To make the effectiveness evaluation, we divide a user's location history into two parts: 1) we select the location

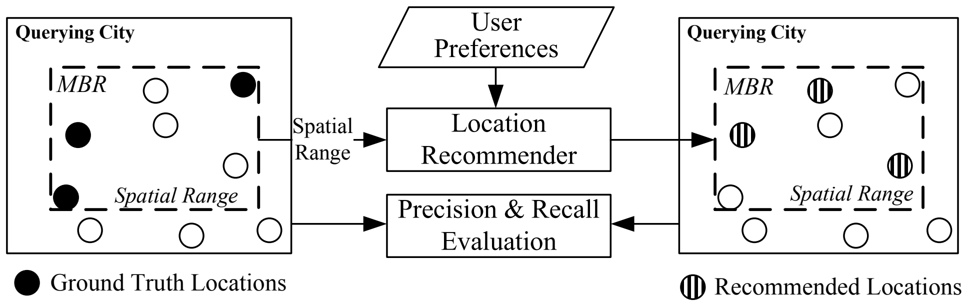


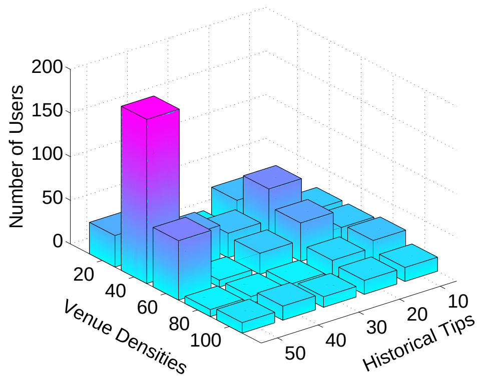
Figure 5.8: Recommendation Effectiveness Evaluation Method.

history generated in a querying city as a test set and 2) we use the rest of the user's location history as a training set for us to learn the user's preferences. We regard the venues that a user has visited in the querying city as the ground truths and match the recommended locations against these venues. The more recommended locations truly visited by a user in the test city, the more effective the recommendation method is. Specifically, as shown in the left part of Figure 5.8, the black dots are the venues the user actually visited, and we regard the minimum bounding box of all the visited venues in the querying city to simulate the geospatial range that would be specified in the user's recommendation request. Remember that our recommendation system is location-aware, i.e., a spatial range is needed here to evaluate the effectiveness. Then, based on the given geospatial range and the user's location history, some venues will be recommended by our system, as illustrated by the striped dots in the right part of Figure 5.8. Based on the ground truth and recommendations, we are able to compute a precision and recall according to Equation 8 and 9.

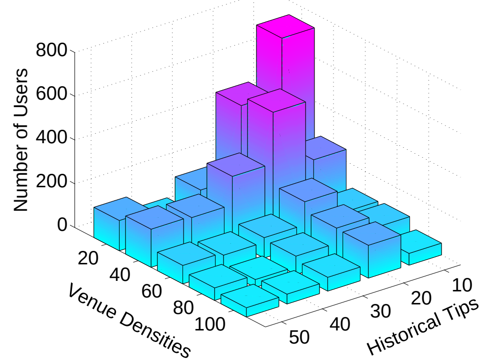
$$precision = \frac{number\ of\ recovered\ ground\ truths}{total\ number\ of\ recommendations} \tag{5.10}$$

$$recall = \frac{number\ of\ recovered\ ground\ truths}{total\ number\ of\ ground\ truths}. \tag{5.11}$$

In fact, this is a very strict evaluation measurement as a user may still like a venue even if the user did not visit the venue. Or, a user has visited a location while the user forgot to leave *tips*. In other words, our method is actually more effective than the number shown in the following experimental results. Meanwhile, the results still reveal the advantages of our method beyond



(a) New Jersey Users in LA.



(b) New Jersey Users in NYC.

Figure 5.9: User Location History Distributions.

baselines from the perspective of a relative comparison.

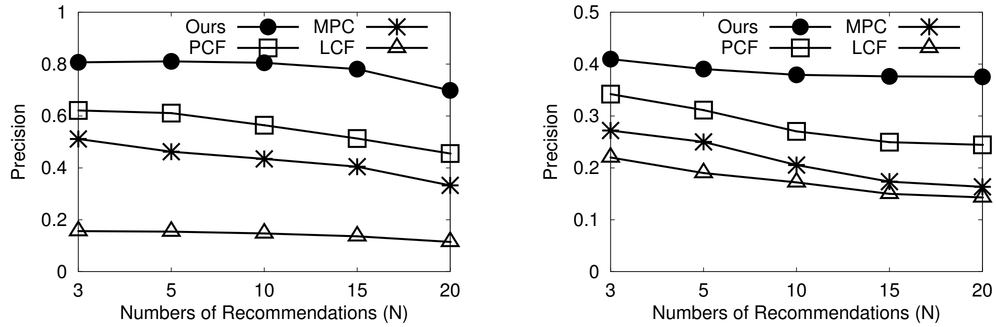
The precision and recall are affected by the following three major factors: 1) the number of requested recommendations N, 2) the scale of a user's location history (i.e., the number of visited locations, including locations outside a querying city), and 3) the density of venues with *tips* in a user's query range (for simplicity termed as venue density). For example, the venue density shown in the left part of Figure 5.8 is 6 (if the size of the bounding box is 1 *mile*^{2}). Therefore, in the rest of the paper, we study the effectiveness of our system changing over these three factors, using the NJ users' data shown in Table 5.1. Figure 5.9 respectively illustrates the distributions of the NJ users in LA and NYC with respect to the scale of location history and the venue density (the number of venues with *tips* per *mile*^{2}).

2) *Recommendation efficiency.* The efficiency of the online recommendation mainly depends on the following two aspects: a) the size of the user-specified geospatial range and b) the number of venues recommended. Therefore, we test the efficiency of our system changing over these two factors. At the same, we explore the benefit the candidate selection component brings to the system.

5.5.2 Experimental Results

Effectiveness of Recommendations

Figure 5.10 and 5.11 show the average precision and recall of different methods varying in the number of recommended locations (N). Clearly, our method outperforms baseline approaches



(a) New Jersey Users in LA. (b) New Jersey Users in NYC.

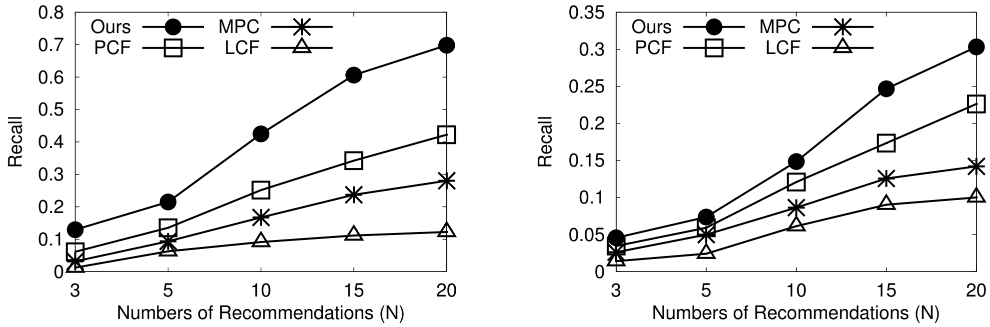
Figure 5.10: Precision w.r.t Recommendation Numbers.

significantly. First, LCF drops behind other three methods, showing the advantage of using location categories to model a user's location history and carrying a location-dependent inference. Second, PCF and our method outperform MPC, justifying the benefit brought by considering social opinions. Third, our method exceeds PCF due to the advantages of WCH, which is more capable of modeling a user's preferences. Finally, our method has a very similar performance between using and without using the candidate select algorithm, as shown in Table 5.3 (we did not plot it on Figure 5.10 and 5.11, as the difference is minor). This is a good result as the candidate selection improves the efficiency of our method (see later results) significantly while having the same (or even better) effectiveness as (or than) using the full set of locations falling in a user-specified geospatial range.

As shown in Figure 5.10 and 5.11, the recall of our method increases quickly though the precision drops slightly as the number of recommendation increases. Our method achieves the best performance when N=15 in LA (*F-measure*=0.771), and N=20 in NYC (*F-measure* =0.385), where F-measure=$2 \times \frac{precision \times recall}{(precision+recall)}$. In addition, the precision in LA is higher than

Method	Precision			Recall		
	N**=5**	N**=10**	N**=20**	N**=5**	N**=10**	N**=20**
Ours	0.80	0.79	0.71	0.21	0.42	0.70
Ours w/o CS	0.81	0.80	0.70	0.21	0.42	0.68

Table 5.3: Comparison of *Ours* & *Ours w/o CS* (NJ users in LA).

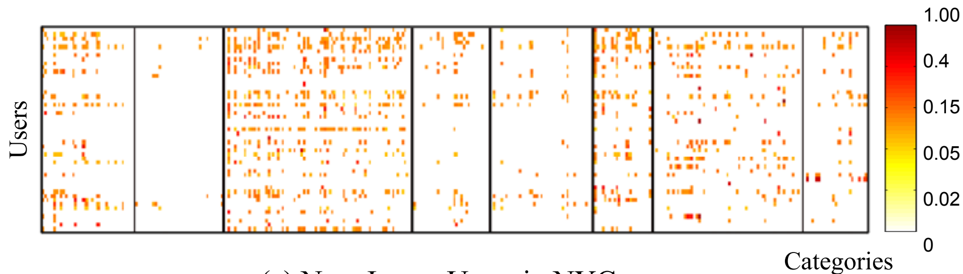


(a) New Jersey Users in LA. (b) New Jersey Users in NYC.

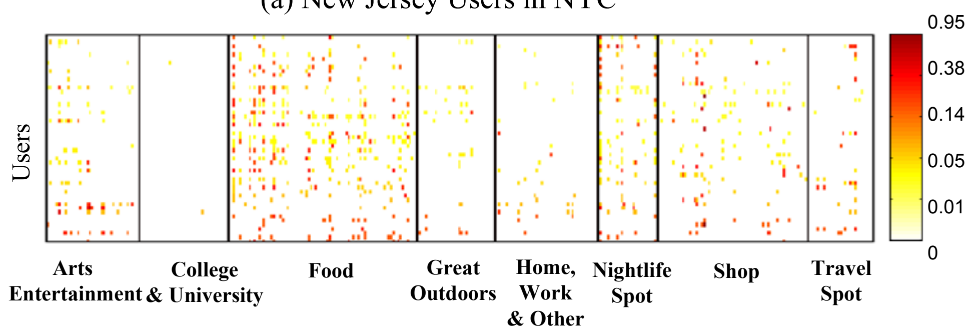
Figure 5.11: Recall w.r.t Recommendation Numbers.

that of NYC though NJ users have more location histories in NYC beyond LA. In other words, the venues to be visited by a user are more predicable when the user travels to a new city. This seems somehow surprising at first glance. However, we found it is true given the following fact: People usually visit some well-known places (e.g., tourist attractions or restaurants introduced in a travel guide book) in a new city to them, while would travel to any venues in a city they are very familiar with (e.g., hometown). This is also one of the reasons leading to a lower recall in NYC. Besides that, NJ users have visited more locations in NYC, causing a bigger denominator in Equation 9 which further reduces the recall. Figure 5.12 further justifies this claim by visualizing the distribution of a user's location history in different categories (in LA and NYC respectively). Here, each row (line) represents a user and each column (line) denotes a category. We select the top-50 users with the largest scale of location history, ranking them from the top to the bottom in the figure. Meanwhile, we group the sub-categories belonging to the same category by a set of separators on the horizontal axis (refer to Figure 2 b)). Clearly, these users' location histories are more focused in LA than in NYC (as NYC is much closer to New Jersey than LA), therefore easy to predict. It is similar to the discovery in [97] that a long-distance travel is more influenced by the social network ties.

To further explore the performance of our method, Figure 5.13 presents the precision of different methods changing over the scale of a user's locations history (where a user requests 10 recommendations, i.e., N=10). As a result, the more locations that a user has visited the more accurate we can model a user's preferences, thereby leading to a better performance. Additionally, the precision of the other three methods increases faster beyond LCF as the number



(a) New Jersey Users in NYC



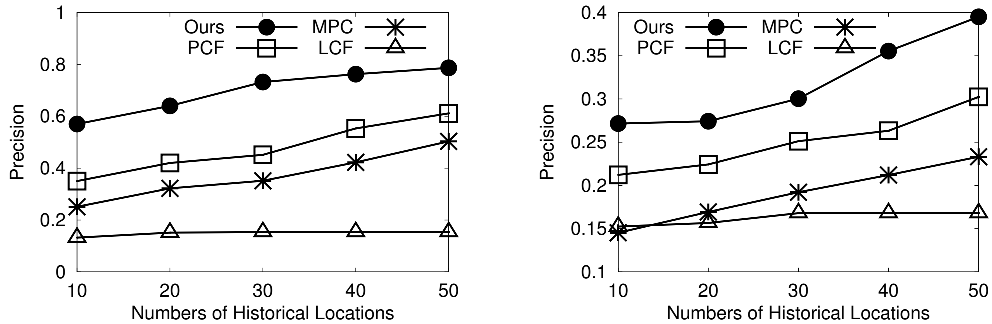
(b) New Jersey Users in LA

Figure 5.12: Category distributions of Top-50 NJ users.

of visited location increases, showing the advantage of location category in dealing with data sparseness problem. Similar to Figure 5.10, the precision in LA is still higher than NYC.

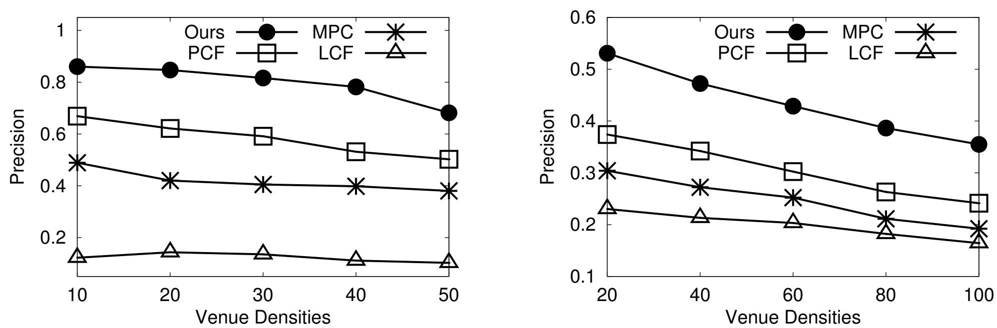
Figure 5.14 plots the precision of different methods changing over the venue density. The results match our intuition that the denser venues located around a user the more location candidates can be recommended. Therefore, the prediction becomes harder and then the precision decreases. Actually, to guarantee the quality of recommendations, our system can help a user smartly determine the number of venues that should be recommended based on the scale of her location history and the venue density around. In this way, a user does not need to do anything when using our system.

Figure 5.15 further studies the user similarity function (using the defined precision and recall criteria), justifying the advantage of each component we defined in Equation 5.8. Here, "Simple" denotes the user similarity solely considering the overlapped nodes between two users' WCHs (i.e., Equation 5.6). "Simple+Level" means the similarity taking into account both the overlapped nodes and the granularity of a WCH (nodes on a deeper level are assigned with a



(a) New Jersey Users in LA. (b) New Jersey Users in NYC.

Figure 5.13: Precision w.r.t Scales of Location Histories.



(a) New Jersey Users in LA. (b) New Jersey Users in NYC.

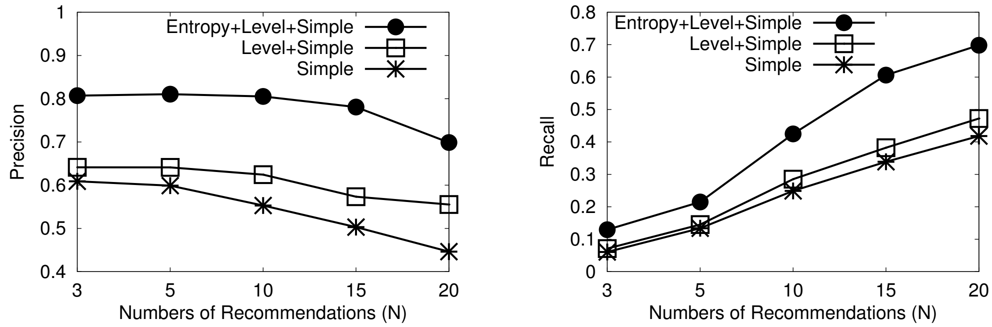
Figure 5.14: Precision w.r.t Venue Densities.

bigger β). Finally, "Simple + Level + Entropy" is the similarity we defined in Equation 5.8. The results show the benefit by adding each component to our similarity function. In addition, the entropy of a WCH brings a significant improvement.

Efficiency of Recommendations

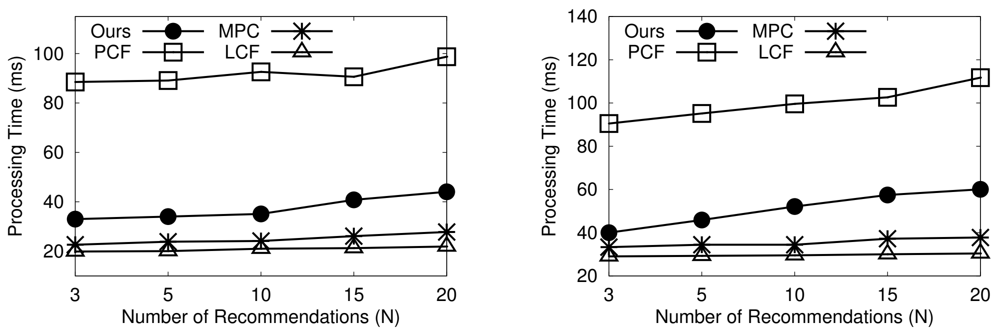
In the efficiency study, we test 200 users in LA and NYC respectively, randomly choosing a location in the city for the user. The experiments were evaluated on a computer running Windows 7 with an Intel Xeon CPU 2.80GHz processor and 24 GB RAM.

Figure 5.16 presents the average online efficiency of different methods varying in the number of recommendations, setting 10 miles as a query range. For example, on average our method can find top-10 location recommendations (that could interest a user most) within a distance of



(a) Precision (NJ users in LA). (b) Recall (NJ users in LA).

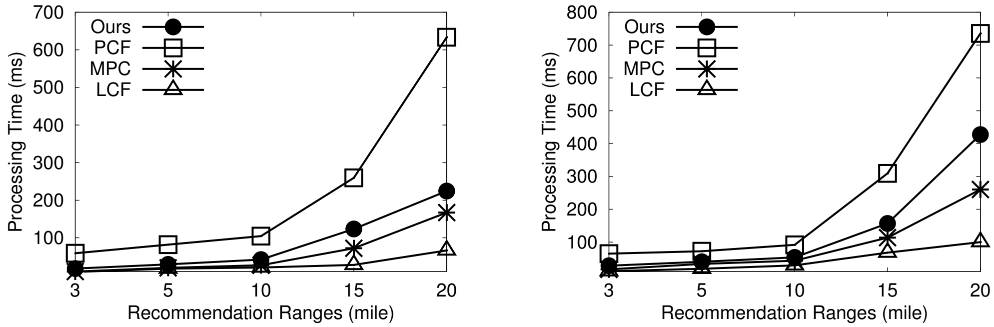
Figure 5.15: Similarity Functions w.r.t Recommendations.



(a) Processing time in LA. (b) Processing time in NYC.

Figure 5.16: Efficiency w.r.t Recommendations (R=10 miles).

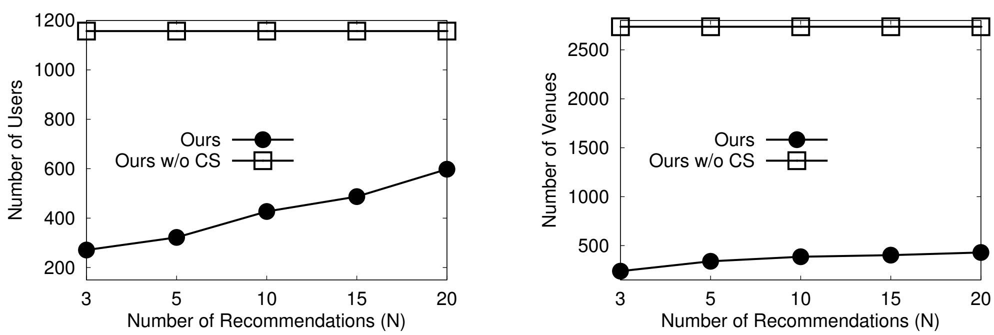
10 mile (to a user's current position) in 40ms in LA and about 60ms in NYC. It is not surprising that our method is slower than MPC which does not consider the location history of other users. LCF achieves the best efficiency because we do not count the time for the CF-based inference (which is supposed to be carried out offline). Theoretically, no method can outperform LCF in efficiency as it only does an online selection (of course, the effectiveness of LCF is the worst among these approaches). But, our method is faster than PCF due to the candidate selection algorithm, and is not significantly slower than MPC and LCF. The processing time only increases slightly as the number of recommendations increases. Additionally, the online recommendation only costs a little bit more time in NYC (than LA) though the venue density in NYC is higher than LA.



(a) Processing time in LA.

(b) Processing time in NYC.

Figure 5.17: Efficiency w.r.t Spatial Ranges (N=10).



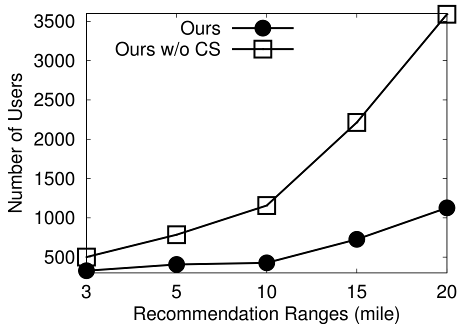
(a) Selected Users in LA.

(b) Candidate Venues in LA.

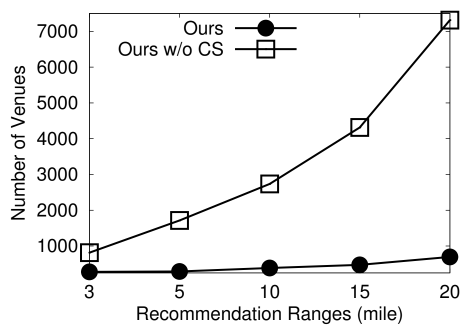
Figure 5.18: Candidates w.r.t Recommendations (R=10 miles).

Figure 5.17 shows the average efficiency of different approaches changing over the geospatial range specified by a user, setting N=10. Intuitively, a larger range will incorporate more location and user candidates, leading to a heavier computational load. But, we find the similar trends as that shown in Figure 5.17 (LCF $>$ MPC $>$ ours $>$ PCF). As people would not request location recommendation far away from them, we only study the efficiency up to 20 miles. Overall, our method is efficient and scalable, besides the effectiveness we have justified before.

To explore the benefit brought by the candidate selection algorithm, we further study the difference between using and without using the candidate selection algorithm. Figure 5.18 a) and b) respectively present the number of users and that of locations chosen for the CF model, varying in number of recommendations (setting range R=10 miles) For instance, our method with candidate selection only employs 1/3 users and 1/5 location candidates for generating 10



(a) Selected Users in LA.



(b) Candidate Venues in LA.

Figure 5.19: Candidates w.r.t Spatial Ranges (N=10).

location recommendations, which is as good as using the full set. In addition, the smaller number of recommendations requested, the more inexperienced users and low quality locations removed. Figure 5.19 a) and b) respectively plot the number of users and that of locations chosen for the CF model, changing over the size of the user-specified geospatial range. As a result, the larger range a user specifies, the more inexperienced users and low quality locations our candidate selection algorithm removes. In short, the candidate selection algorithm improves the efficiency of our system significantly while maintaining the effectiveness.

5.6 Summary

This paper presents a location-based and preference-aware recommender system, which provides a user with location recommendations around the specified geo-position based on 1) the user's personal preferences learnt from her location history and 2) social opinions mined from the *local experts* who could share similar interests. This recommender system can facilitate people's travel not only near their living areas but also to a city that is new to them (even if they have not visited any places there). By taking advantage of the category information of a user's location history, our system overcomes the data sparsity problem in the original user-location matrix. We evaluated our system using extensive experiments based on a real data set (221,128 *tips* generated by 49,062 users in NYC and 104,478 *tips* generated by 31,544 users in Los Angeles) collected from Foursquare. According to the experimental results, our approach significantly outperforms some major location recommendation methods (MPC, LCF, and PCF) in effectiveness (measured by precision and recall). The results also justify each component proposed in our system, e.g., taking into account location history of others, category-hierarchy based preference modeling, user similarity computing, and CF-based inference. Meanwhile, the proposed candidate selection algorithm improves the efficiency of our approach tremendously while maintaining the effectiveness, enabling an online recommendation scenario. In general, our system can provide 10 quality location recommendations within a 10-mile spatial range within 60ms. In the future, we are going to incorporate the temporal and weather features into the recommendation system.

Chapter 6

Conclusion and Discussion

6.1 Conclusion

This dissertation presents my work on building socialized location-based services. The dissertation first presents the background and the need for the socialized location-based services. Then, I describe unique challenges in enabling the socialized information in a traditional location-based service, from both the system and user's perspective. The main work of this dissertation focuses on improving the system efficiency and effectiveness on different socialized location-based applications, as 1) location-aware news feed, 2) location-aware news ranking, and 3) location-aware recommender. In Chapter 2, we discuss the state-of-art techniques used in the traditional news feed, news ranking and recommendations.

In Chapter 3, we proposed a new service that enables the users to enjoy the spatial-aware news feed from her subscribed agencies or friends based on the current location. We propose three different approaches to complete the task, as: 1) spatial pull approach, 2) spatial push approach and 3) shared push approach. Furthermore, we design a smart decision algorithm that takes the cost model and user's various activity patterns into the consideration to select the best approach for each user-friend pair. In that way, we can guarantee the response delay for the user without overwhelming the system with the massive message updates.

In Chapter 4, we propose a location-aware news ranking service, which further extends the GeoFeed by providing a personalized and spatio-temporal aware ranking. In the GeoRank, we design two main components as: 1) GeoRank query processor, which uses some pre-computed statistics to significantly prune the unnecessary computations in retrieving the top-k query answers from a large number of user's subscribed agencies/friends. and 1) GeoRank message updater, which efficiently handles the spatial message updates from the user's friends and changing the statistics maintained in the system to facilitate the pruning steps in the query processor.

In Chapter 5, we propose a novel location-aware recommender that could effectively provide the location suggestions for the users when they are in a new area. Our system first builds a user's preference by mining the category information within her location history. On the other side, we use HITS algorithm to extract the local experts based on a region and a location category. Finally, the recommendation is generated using the suggestions from the local experts and the querying user's preferred location category.

Besides the work in providing the socialized location-based services, another line of my Ph.D research work is to design efficient query processing algorithms on spatial networks. For example, we propose an efficient algorithm [98] to discover the k nearest neighbors in a road network from a region contains a set of road segments. Moreover, we have design and implement a real system, MNTG [99, 100] [1] , an extensive road traffic generator that enable the users to generate road network-based traffic via a web-interface using different traffic models. Finally, we also build a system, i.e., iRoad [101], for efficiently processing the predictive queries in the road networks.

6.2 Future Works

In this section, we outline some major open issues related to socialized location-based services that should be addressed in the future.

Spatial diversity in query results. Currently, the results of the news feed from the proposed system, e.g., GeoFeed and GeoRank are strictly based on the user's current location and time, where the more close items will be given higher priority. However, in the case when a user get to a new city, she may see a lot of similar news items that very close to her current location, while she may interested in the things happening throughout the city. To this end, we want to introduce the spatial diversity to the query answers to improve the effectiveness in the news feed result. Although, many diversification techniques, e.g.,[102, 103, 104, 105], have been proposed to enable the query diversity, none of them is focus on the spatial aspect. To enable

[1] http://mntg.cs.umn.edu

the spatial diversity in the news ranking services, we expect to develop a more efficient and effective news feed services for the users.

Location privacy issue in query results. Location privacy has always been an issue since the availability of the location-based services [106, 107]. To realize the socialized location-based service services, we will need to get more location updates from the users. Also in order to study a user's personal preference, we need to study the history of an individual. Both of the above applications may raise concerns for location privacy issues [34, 35, 108]. To this end, a more sophisticated system and algorithms are needed to handle the query processing, location update and data mining processing in the socialized location-based services.

References

[1] Tech Pluto. Core Characteristics of Web 2.0 Services. http://www.techpluto.com/web-20-services/.

[2] Library 2.0. http://www.library20.org/.

[3] Travel 2.0. http://travel2dot0.com/.

[4] Government 2.0 Summit. Washington D.C., USA, September, 2010. http://www.gov2summit.com/gov2010.

[5] Mark Sedra. Revolution 2.0: democracy promotion in the age of social media. The Global and Mail. *http://www.theglobeandmail.com/news/opinions/opinion/revolution-20-democracy-promotion-in-the-age-of-socialmedia/article1891015/*, 2011.

[6] Justin J Levandoski, Mohamed F Mokbel, and Mohamed E Khalefa. Caredb: A context and preference-aware location-based database system. *Proceedings of the VLDB Endowment*, 3(1-2):1529–1532, 2010.

[7] Jie Bao, Mohamed F. Mokbel, and Chi-Yin Chow. GeoFeed: A Location-aware News Feed System. In *ICDE*, pages 54–65, 2012.

[8] Waldo R Tobler. A computer movie simulating urban growth in the detroit region. *Economic geography*, pages 234–240, 1970.

[9] Quannan Li, Yu Zheng, Xing Xie, Yukun Chen, Wenyu Liu, and Wei-Ying Ma. Mining user similarity based on location history. In *ACM SIGSPATIAL*, page 34. ACM, 2008.

[10] Xiangye Xiao, Yu Zheng, Qiong Luo, and Xing Xie. Finding similar users using category-based location history. In *SIGSPATIAL*, pages 442–445. ACM, 2010.

[11] Peter DeScioli, Robert Kurzban, Elizabeth N Koch, and David Liben-Nowell. Best friends alliances, friend ranking, and the myspace social network. *Perspectives on Psychological Science*, 6(1):6–8, 2011.

[12] Justin J Levandoski, Mohamed Sarwat, Ahmed Eldawy, and Mohamed F Mokbel. LARS: A Location-Aware Recommender System. In *ICDE*, pages 450–461, 2012.

[13] Mao Ye, Peifeng Yin, Wang-Chien Lee, and Dik-Lun Lee. Exploiting geographical influence for collaborative point-of-interest recommendation. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pages 325–334. ACM, 2011.

[14] Vincent Wenchen Zheng, Yu Zheng, and Qiang Yang. Joint learning user's activities and profiles from gps data. In *Proceedings of the 2009 International Workshop on Location Based Social Networks*, pages 17–20. ACM, 2009d.

[15] Karen P Tang, Jialiu Lin, Jason I Hong, Daniel P Siewiorek, and Norman Sadeh. Rethinking location sharing: exploring the implications of social-driven vs. purpose-driven location sharing. In *Proceedings of the 12th ACM international conference on Ubiquitous computing*, pages 85–94. ACM, 2010.

[16] Nathan Eagle and Alex Pentland. Reality mining: sensing complex social systems. *Personal and Ubiquitous Computing*, 10(4):255–268, 2006.

[17] Nathan Eagle, Alex Sandy Pentland, and David Lazer. Inferring friendship network structure by using mobile phone data. *Proceedings of the National Academy of Sciences*, 106(36):15274, 2009.

[18] Y. Zheng and X. Zhou. *Computing with Spatial Trajectories*. Springer, 2011.

[19] Mohamed Sarwat, Jie Bao, Ahemd Eldawy, Justin J. Levandoski, Amr Magdy, and Mohamed F. Mokbel. Sindbad: A Location-based Social Networking System. In *SIGMOD*, pages 649–652, 2012.

[20] Badrish Chandramouli and Jun Yang. End-to-end support for joins in large-scale publish/subscribe systems. *PVLDB*, 1(1), 2008.

[21] Badrish Chandramouli, Jun Yang, Pankaj K. Agarwal, Albert Yu, and Ying Zheng. ProSem: scalable wide-area publish/subscribe. In *SIGMOD Conference*, 2008.

[22] Yongluan Zhou, Ali Salehi, and Karl Aberer. Scalable delivery of stream query results. *PVLDB*, 2(1), 2009.

[23] Adam Silberstein, Jeff Terrace, Brian F. Cooper, and Raghu Ramakrishnan. Feeding Frenzy: Selectively Materializing User's Event Feed. In *SIGMOD*, 2010.

[24] Facebook. http://www.facebook.com/.

[25] Renren. http://www.renren.com/.

[26] Sina Weibo. http://t.sina.com.cn/.

[27] Loopt. http://www.loopt.com.

[28] Google Buzz Moblie. http://www.google.com/buzz.

[29] FourSquare. http://www.foursquare.com/.

[30] Twinkle. http://tapulous.com/twinkle/.

[31] Lauri Aalto, Nicklas Göthlin, Jani Korhonen, and Timo Ojala. Bluetooth and wap push based location-aware mobile advertising system. In *MobiSys*, 2004.

[32] Ying Cai and Toby Xu. Design, analysis, and implementation of a large-scale real-time location-based information sharing system. In *MobiSys*, 2008.

[33] Dario Freni, Carmen Ruiz Vicente, Sergio Mascetti, Claudio Bettini, and Christian S. Jensen. Preserving location and absence privacy in geo-social networks. In *CIKM*, 2010.

[34] Ali Khoshgozaran and Cyrus Shahabi. Private Buddy Search: Enabling Private Spatial Queries in Social Networks. In *12th IEEE International Conference on Computational Science and Engineering*, 2009.

[35] Laurynas Siksnys, Jeppe Thomsen, Simonas Saltenis, and Man Lung Yiu. Private and Flexible Proximity Detection In Mobile Social Networks. In *MDM*, 2010.

[36] Laurynas Siksnys, Jeppe R. Thomsen, Simonas Saltenis, Man Lung Yiu, and Ove Andersen. A Location Privacy Aware Friend Locator. In *SSTD*, 2009.

[37] Moon-Hee Park, Jin-Hyuk Hong, and Sung-Bae Cho. Location-based recommendation system using bayesian user's preference model in mobile devices. *Ubiquitous Intelligence and Computing*, pages 1130–1139, 2007.

[38] Mao Ye, Peifeng Yin, and Wang-Chien Lee. Location recommendation for location-based social networks. In *SIGSPATAIL*, pages 458–461. ACM, 2010.

[39] Vincent W Zheng, Yu Zheng, Xing Xie, and Qiang Yang. Collaborative location and activity recommendations with gps history data. In *WWW*, pages 1029–1038. ACM, 2010.

[40] Yu Zheng, Yukun Chen, Xing Xie, and Wei-Ying Ma. GeoLife2.0: A Location-Based Social Networking Service. In *MDM*, 2009.

[41] Gianna M. Del Corso, Antonio Gull, and Francesco Romani. Ranking a stream of news. In *WWW*, pages 97–106, 2005.

[42] Lihong Li, Wei Chu, John Langford, and Robert E. Schapire. A Contextual-Bandit Approach to Personalized News Article Recommendation. In *WWW*, pages 661–670, 2010.

[43] Abhinandan S Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram. Google news personalization: scalable online collaborative filtering. In *Proceedings of the 16th international conference on World Wide Web*, pages 271–280. ACM, 2007.

[44] Jiahui Liu, Peter Dolan, Elin Ronby, and Pedersen. Personalized News Recommendation Based on Click. In *IUI*, pages 31–40, 2010.

[45] Benjamin E. Teitler, Michael D. Lieberman, Daniele Panozzo, Jagan Sankaranarayanan, Hanan Samet, and Jon Sperling. NewsStand: a new view on news. In *SIGSPATIAL*, page 18, 2008.

[46] Gianluca Quercini, Hanan Samet, Jagan Sankaranarayanan, and Michael D. Lieberman. Determining the spatial reader scopes of news sources using local lexicons. In *SIGSPATIAL*, pages 43–52, 2010.

[47] Wenjian Xu, Chi-Yin Chow, Man Lung Yiu, Qing Li, and Chung Keung Poon. MobiFeed: A Location-Aware News Feed System for Mobile Users. In *SIGSPATIAL*, 2012.

[48] Ihab F. Ilyas, George Beskales, and Mohamed A. Soliman. A Survey of Top-k Query Processing Techniques in Relational Database Systems. *ACM Computing Surveys*, 40(4):11, 2008.

[49] Ulrich Güntzer, Wolf-Tilo Balke, and Werner Kießling. Optimizing Multi-Feature Queries for Image Databases. In *VLDB*, pages 419–428, 2000.

[50] Surya Nepal and M.V. Ramakrishna. Query Processing Issues in Image (Multimedia) Databases. In *ICDE*, pages 22–29, 1999.

[51] Jiaheng Lu, Pierre Senellart, Chunbin Lin, Xiaoyong Du, Shan Wang, , and Xinxing Chen. Optimal Top-k Generation of Attribute Combinations based on Ranked Lists. In *SIGMOD*, pages 409–420, 2012.

[52] Pei Cao and Zhe Wang. Efficient Top-k Query Calculation in Distributed Networks. In *PODC*, pages 206–215. ACM, 2004.

[53] Albert Yu, Pankaj K. Agarwal, and Jun Yang. Processing a Large Number of Continuous Preference Top-k Queries. In *SIGMOD*, pages 397–408, 2012.

[54] Yanhua Li, Zhi-Li Zhang, and Jie Bao. Mutual or Unrequited Love: Identifying Stable Clusters in Social Networks with Uni- and Bi-directional Links. In *WAW*, pages 113–125, 2012.

[55] Jan Chomicki, Parke Godfrey, Jarek Gryz, and Dongming Liang. Skyline with presorting. In *ICDE*, pages 717–719, 2003.

[56] Wolf-Tilo Balke and Ulrich Guntzer. Multi-objective Query Processing for Database Systems. In *VLDB*, pages 936–947, 2004.

[57] Man Lung Yiu and Nikos Mamoulis. Efficient Processing of Top-k Dominating Queries on Multi-Dimensional Data. In *VLDB*, pages 483–494, 2007.

[58] Gisli R. Hjaltason and Hanan Samet. Distance Browsing in Spatial Databases. *ACM TODS*, 24(2):265–318, 1999.

[59] Mehdi Sharifzadeh and Cyrus Shahabi. The Spatial Skyline Queries. In *VLDB*, pages 751–762, 2006.

[60] Man Lung Yiu, Hua Lu, Nikos Mamoulis, and Michail Vaitis. Ranking spatial data by quality preferences. *IEEE Transactions on Knowledge and Data Engineering*, 2010.

[61] Jie Bao, Yu Zheng, and Mohamed F. Mokbel. Location-based and preference-aware recommendation using sparse geo-social networking data. In *SIGSPATIAL*, pages 184–195, 2012.

[62] Yu Zheng, Lizhu Zhang, Xing Xie, and Wei-Ying Ma. Mining interesting locations and travel sequences from gps trajectories. In *WWW*, pages 791–800. ACM, 2009.

[63] Xin Cao, Gao Cong, and Christian S Jensen. Mining significant semantic locations from gps data. *Proceedings of the VLDB Endowment*, 3(1-2):1009–1020, 2010.

[64] Kazuki Kodama, Yuichi Iijima, Xi Guo, and Yoshiharu Ishikawa. Skyline queries based on user locations and preferences for making location-based recommendations. In *GIS-LBSN*, 2009.

[65] Chi-Yin Chow, Jie Bao, and Mohamed F. Mokbel. Towards Location-based Social Networking Services. In *The 2nd Workshop on LBSN*, 2010.

[66] Yu Zheng, Lizhu Zhang, Zhengxin Ma, Xing Xie, and Wei-Ying Ma. Recommending friends and locations based on individual location history. *ACM Transactions on the Web*, 5(1):5, 2011.

[67] Yu Zheng and Xing Xie. Learning travel recommendations from user-generated GPS traces. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(1):2, 2011.

[68] Hao Ma, Haixuan Yang, Michael R Lyu, and Irwin King. Sorec: social recommendation using probabilistic matrix factorization. In *CIKM*, pages 931–940. ACM, 2008.

[69] A.I. Schein, A. Popescul, L.H. Ungar, and D.M. Pennock. Methods and metrics for cold-start recommendations. In *SIGIR*, pages 253–260. ACM, 2002.

[70] Hyung Jun Ahn. A new similarity measure for collaborative filtering to alleviate the new user cold-starting problem. *Information Sciences*, 178(1):37–51, 2008.

[71] Twitter. http://www.twitter.com/.

[72] MyYahoo! http://my.yahoo.com/.

[73] iGoogle. http://www.google.com/ig.

[74] Reynold Cheng, Yuni Xia, Sunil Prabhakar, and Rahul Shah. Change Tolerant Indexing for Constantly Evolving Data. In *ICDE*, 2005.

[75] Mohamed F. Mokbel, Xiaopeng Xiong, and Walid G. Aref. SINA: Scalable Incremental Processing of Continuous Queries in Spatio-temporal Databases. In *SIGMOD*, 2004.

[76] Sunil Prabhakar, Yuni Xia, Dmitri V. Kalashnikov, Walid G. Aref, and Susanne E. Hambrusch. Query Indexing and Velocity Constrained Indexing: Scalable Techniques for Continuous Queries on Moving Objects. *IEEE Trans. Computers*, 51(10), 2002.

[77] PostgreSQL . www.postgresql.org.

[78] Twitter Search API. http://search.twitter.com.

[79] TinyGeoCoder. http://tinygeocoder.com/.

[80] Google GeoCoding. http://code.google.com/apis/maps/documentation/geocoding.

[81] Facebook Statistics. http://www.facebook.com/press/info.php?statistics, 2010.

[82] George Kingsley Zipf. *Human Behaviour and the Principle of Least Effort: an Introduction to Human Ecology*. Addison-Wesley, 1949.

[83] Yu Zheng. Location-based social networks: Users. In *Computing with Spatial Trajectories, Zheng, Yu and Zhou, Xiaofang, Ed.* Springer, 2011.

[84] Yu Zheng, Xing Xie, and Wei-Ying Ma. Geolife: A collaborative social networking service among user, location and trajectory. *IEEE Data Engineering Bulletin*, 33(2):32–40, 2010.

[85] Mohamed Mokbel, Jie Bao, Ahmed Eldawy, Justin Levandoski, and Mohamed Sarwat. Personalization, Socialization, and Recommendations in Location-based Services 2.0. In *PersDB*. VLDB, 2011.

[86] Anastasios Noulas, Salvatore Scellato, Cecilia Mascolo, and Massimiliano Pontil. Exploiting semantic annotations for clustering geographic areas and users in location-based social networks. In *Fifth International AAAI Conference on Weblogs and Social Media*, 2011.

[87] Tzvetan Horozov, Nitya Narasimhan, and Venu Vasudevan. Using location for personalized poi recommendations in mobile environments. In *SAINT*, pages 124–129, 2006.

[88] Jonathan L Herlocker, Joseph A Konstan, Al Borchers, and John Riedl. An algorithmic framework for performing collaborative filtering. In *SIGIR*, pages 230–237. ACM, 1999.

[89] J.A. Konstan, B.N. Miller, D. Maltz, J.L. Herlocker, L.R. Gordon, and J. Riedl. Grouplens: applying collaborative filtering to usenet news. *Communications of the ACM*, 40(3):77–87, 1997.

[90] Salvatore Scellato, Anastasios Noulas, Renaud Lambiotte, and Cecilia Mascolo. Socio-spatial properties of online location-based social networks. *Proceedings of ICWSM*, 11, 2011.

[91] Xavier Amatriain, Neal Lathia, Josep M Pujol, Haewoon Kwak, and Nuria Oliver. The wisdom of the few: a collaborative filtering approach based on expert opinions from the web. In *ACM SIGIR*, pages 532–539. ACM, 2009.

[92] C-H Lee, Y-H Kim, and P-K Rhee. Web personalization expert with combining collaborative filtering and association rule mining technique. *Expert Systems with Applications*, 21(3):131–137, 2001.

[93] Stephen Robertson. Understanding inverse document frequency: on theoretical arguments for idf. *Journal of Documentation*, 60(5):503–520, 2004.

[94] Soumen Chakrabarti, Byron Dom, Prabhakar Raghavan, Sridhar Rajagopalan, David Gibson, and Jon Kleinberg. Automatic resource compilation by analyzing hyperlink structure and associated text. *Computer Networks and ISDN Systems*, 30(1-7):65–74, 1998.

[95] Jon. M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM (JACM)*, 46(5):604–632, 1999.

[96] Justin Cranshaw, Eran Toch, Jason Hong, Aniket Kittur, and Norman Sadeh. Bridging the gap between physical location and online social networks. In *Ubicomp*, pages 119–128. ACM, 2010.

[97] Eunjoon Cho, Seth A Myers, and Jure Leskovec. Friendship and mobility: user movement in location-based social networks. In *Proceedings of the 17th ACM SIGKDD*, pages 1082–1090. ACM, 2011.

[98] Jie Bao, Chi-Yin Chow, Mohamed F. Mokbel, and Wei-Shinn Ku. Efficient Evaluation of k-Range Nearest Neighbor Queries in Road Networks. In *MDM*, 2010.

[99] Mohamed F. Mokbel, Louai Alarabi, Jie Bao, Amr Magdy Ahmed Eldawy, Mohamed Sarwat, Ethan Waytas, and Steven Yackel. MNTG: An Extensible Web-based Traffic Generator. In *SSTD*, 2013.

[100] Mohamed F. Mokbel, Louai Alarabi, Jie Bao, Amr Magdy Ahmed Eldawy, Mohamed Sarwat, Ethan Waytas, and Steven Yackel. A Demonstration of MNTG - A Web-based Road Network Traffic Generator. In *ICDE*, 2014.

[101] Abdeltawab M. Hendawi, Jie Bao, and Mohamed F. Mokbel. iRoad: A Framework For Scalable Predictive Query Processing On Road Networks. In *PVLDB*, 2013.

[102] Anoop Jain, Parag Sarda, and Jayant R Haritsa. Providing diversity in k-nearest neighbor query results. In *Advances in Knowledge Discovery and Data Mining*, pages 404–413. Springer, 2004.

[103] Paul Clough, Mark Sanderson, Murad Abouammoh, Sergio Navarro, and Monica Paramita. Multiple approaches to analysing query diversity. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 734–735. ACM, 2009.

[104] Marina Drosou and Evaggelia Pitoura. Search result diversification. *ACM SIGMOD Record*, 39(1):41–47, 2010.

[105] Lu Li and Chee-Yong Chan. Efficient indexing for diverse query results. *Proceedings of the VLDB Endowment*, 6(9):745–756, 2013.

[106] Mohamed F Mokbel, Chi-Yin Chow, and Walid G Aref. The new casper: query processing for location services without compromising privacy. In *Proceedings of the 32nd international conference on Very large data bases*, pages 763–774. VLDB Endowment, 2006.

[107] Chi-Yin Chow, Mohamed F Mokbel, Jie Bao, and Xuan Liu. Query-aware location anonymization for road networks. *GeoInformatica*, 15(3):571–607, 2011.

[108] Xin Lin, Haibo Hu, Hong Ping Li, Jianliang Xu, and Byron Choi. Private proximity detection and monitoring with vicinity regions. In *Proceedings of the 12th International ACM Workshop on Data Engineering for Wireless and Mobile Acess*, pages 5–12. ACM, 2013.